

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of: Jerome L. Schneider
U.S. Patent No.: 8,856,505 Attorney Docket No.: 54971-0004IP1
Issue Date: Oct 7, 2014
Appl. Serial No.: 13/460,655
Filing Date: April 30, 2012
Title: MALWARE MANAGEMENT THROUGH KERNEL
DETECTION DURING A BOOT SEQUENCE

Mail Stop Patent Board

Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

**PETITION FOR *INTER PARTES* REVIEW OF UNITED STATES PATENT
NO. 8,856,505 PURSUANT TO 35 U.S.C. §§ 311–319, 37 C.F.R. § 42**

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	REQUIREMENTS FOR IPR—37 C.F.R. §42.104.....	1
	A. Grounds for Standing Under 37 C.F.R. § 42.104(a).....	1
	B. Challenge and Relief Requested Under 37 C.F.R. §42.104(b)	1
III.	THE '505 PATENT	2
	A. Brief Description.....	2
	B. Prosecution History.....	3
	C. Level of Ordinary Skill	3
	D. Claim Construction	4
IV.	THE CHALLENGED CLAIMS ARE UNPATENTABLE.....	4
	A. [GROUND 1] – CLAIMS 1-20 WOULD HAVE BEEN OBVIOUS OVER FIELD, GLICK, AND SHUKLA	5
	1. Field (EX1004).....	5
	2. Glick (EX1007)	8
	3. Shukla (EXS. 1005, 1006).....	9
	4. Motivation to combine Field, Glick, and Shukla	13
	5. Element-by-Element Explanation showing the combination of Field, Glick, and Shukla renders challenged claims obvious	21
V.	PTAB DISCRETION SHOULD NOT PRECLUDE INSTITUTION.....	70
	A. The <i>Fintiv</i> Factors Favor Institution—§314(a)	71
	1. Factor 1	71
	2. Factor 2	71
	3. Factor 3	74
	4. Factor 4	74
	5. Factor 5	75
	6. Factor 6	75
	B. § 325(d) Favors Institution	75
VI.	FEES	76
VII.	CONCLUSION.....	76
VIII.	MANDATORY NOTICES UNDER 37 C.F.R § 42.8(a)(1).....	76
	A. Real Party-In-Interest Under 37 C.F.R. § 42.8(b)(1).....	76
	B. Related Matters Under 37 C.F.R. § 42.8(b)(2)	76
	C. Lead And Back-Up Counsel Under 37 C.F.R. § 42.8(b)(3).....	77
	D. Service Information	77

EXHIBITS

EX1001	U.S. Patent No. 8,856,505 (“the ’505 Patent”)
EX1002	Excerpts from the Prosecution History of the ’505 Patent (“the Prosecution History”)
EX1003	Declaration of Dr. Markus Jakobsson
EX1004	U.S. Pat. App. No. 2008/0005797 A1 (“Field”)
EX1005	U.S. Pat. App. No. 2008/0016339 A1 (“Shukla”)
EX1006	U.S. Provisional Application 60/806,143 (“ ’143 Application”)
EX1007	U.S. Pat. No. 8,239,947 B1 (“Glick”)
EX1008	[RESERVED]
EX1009	U.S. Pat. No. 7,784,098 B1 (“Fan”)
EX1010	U.S. Pat. No. 7,673,341 B2 (“Kramer”)
EX1011	Silberschatz, Operating System Concepts (2004)
EX1012	<i>Webroot, Inc. v. CrowdStrike, Inc.</i> , Case No. 6:22-cv-00243-ADA ¹ , Docket No. 252, “Order Granting Fifth Amended Scheduling Order” (WDTX Mar. 29, 2023)

¹ The case number for *Webroot, Inc. v. CrowdStrike, Inc.* is 6:22-CV-00241-ADA-DTG, but the case was consolidated for all pretrial issues (except venue and disqualification) with 6:22-CV-00243-ADA, and the joint filings are available under the docket of 6:22-CV-00243-ADA.

- EX1013 *Webroot, Inc. v. CrowdStrike, Inc.*, Case No. 6:22-cv-00243-
ADA, Docket No. 350, “Order Resetting Markman Hearing”
(WDTX Jun. 8, 2023)
- EX1014 *Memorandum: Interim Procedure for Discretionary Denials in*
AIA Post-Grant Proceedings with Parallel District Court
Litigation (June 21, 2022) (“Director’s Memo”)
- EX1015 U.S. District Courts—Median Time Intervals From Filing to
Disposition of Civil Cases Terminated, by District and Method
of Disposition, During the 12-Month Period Ending March 31,
2023

CLAIM LISTING

Limitation	Claim Language
1[pre]	A method for managing pestware on a computer comprising:
1[a]	monitoring events during a boot sequence of the computer;
1[b]	managing pestware-related events during a first period in a boot sequence of the computer, the first period in the boot sequence occurring before the computer becomes configured to run native applications, before a subsystem of an operating system is loaded, and after a kernel is loaded;
1[c]	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;
1[d]	generating, in response to the monitoring, a record of events, the record of events including the pestware-related events;
1[e]	analyzing the record of events so as to identify the pestware-related events
1[f]	modifying the set of behavior rules so as to prevent the pestware related events; and
1[g]	scanning data in a registry of the computer for pestware during the second period in the boot sequence.
2[a]	The method of claim 1, further comprising launching, after an operating system is initiated, a pestware management engine; and
2[b]	wherein the set of behavior rules includes behavior rules compiled by the pestware management engine.
[3]	The method of claim 1, wherein the record of events includes information selected from a group consisting of: process identification information, file identification information, and hook generation information.
[4]	The method of claim 1, further comprising: managing pestware-related events after the computer becomes configured to run native applications.
[5]	The method of claim 1, wherein the monitoring includes monitoring the boot sequence while boot drivers are initiated.

Limitation	Claim Language
6[pre]	A system for managing pestware on a computer comprising:
6[a]	a processor;
6[b]	a memory comprising:
6[b.i]	a first instruction set configured to monitor events during a boot sequence of the computer;
6[b.ii]	a second instruction set configured to manage pestware-related events during first period in a boot sequence of the computer, the first period in the boot sequence occurring before the computer becomes configured to run native applications, before a WIN32 subsystem is loaded, and after a kernel is loaded;
6[b.iii]	a third instruction set configured to manage 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;
6[b.iv]	a fourth instruction set configured to generate, in response to the monitoring, a record of events, the record of events including the pestware-related events;
6[b.v]	a fifth instruction set configured to analyze the record of events so as to identify the pestware-related events; and
6[b.vi]	a sixth instruction set configured to modify the set of behavior rules so as to prevent the pestware related events; and
6[b.vii]	a seventh instruction set configured to scan data in a registry of the computer for pestware during the second period in the boot sequence.
[7]	The system of claim 6, wherein the memory further comprises an eighth instruction set configured to launch, after an operating system is initiated, a pestware management engine; and wherein the set of behavior rules includes behavior rules compiled by the pestware management engine.
[8]	The system of claim 1, wherein the record of events includes information selected from a group consisting of: process identification information, file identification information, and hook generation information.

Limitation	Claim Language
[9]	The system of claim 6, wherein the memory further comprises: a ninth instruction set configured to manage pestware-related events after the computer becomes configured to run native applications.
[10]	The system of claim 6, wherein the first instruction set is further configured to monitor the boot sequence while boot drivers are initiated.
11[pre]	A non-transitory tangible computer-readable storage medium comprising program instructions for:
11[a]	monitoring events during a boot sequence of a computer;
11[b]	managing pestware-related events during a first period in a boot sequence of the computer, the first period in the boot sequence occurring before the computer becomes configured to run native applications, before a WIN32 subsystem is loaded, and after a kernel is loaded;
11[c]	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;
11[d]	generating, in response to the monitoring, a record of events, the record of events including the pestware-related events;
11[e]	analyzing the record of events so as to identify the pestware-related events;
11[f]	modifying the set of behavior rules so as to prevent the pestware related events; and
11[g]	scanning data in a registry of the computer for pestware during the second period in the boot sequence.
[12]	The non-transitory tangible computer-readable storage medium of claim 11 further comprising program instructions for launching, after an operating system is initiated, a pestware management engine; and wherein the set of behavior rules includes behavior rules compiled by the pestware management engine.

Limitation	Claim Language
[13]	The non-transitory tangible computer-readable storage medium of claim 11, wherein the record of events includes information selected from a group consisting of: process identification information, file identification information, and hook generation information.
[14]	The non-transitory tangible computer-readable storage medium of claim 11 further comprising program instructions for: managing pestware-related events after the computer becomes configured to run native applications.
[15]	The non-transitory tangible computer-readable storage medium of claim 11, wherein the monitoring includes monitoring the boot sequence while boot drivers are initiated.
[16]	The method of claim 2, further comprising modifying, by the pestware management engine, the behavior rules utilized by a kernel-level monitor, wherein the kernel-level monitor manages pestware-related events in the first and second periods in the boot sequence.
[17]	The method of claim 1, further comprising modifying the set of behavior rules based on the scanned registry so as to prevent the pestware-related events.
[18]	The method of claim 1, wherein the pestware-related events include events associated with at least one of loading pestware, executing pestware, and one or more behaviors of pestware.
[19]	The system of claim 6, wherein the pestware-related events include events associated with at least one of loading pestware, executing pestware, and one or more behaviors of pestware.
[20]	The non-transitory tangible computer-readable storage medium of claim 11, wherein the pestware-related events include events associated with at least one of loading pestware, executing pestware, and one or more behaviors of pestware.

I. INTRODUCTION

CrowdStrike, Inc. (“CrowdStrike” or “Petitioner”) petitions for *Inter Partes* Review (“IPR”) of claims 1-20 (“the Challenged Claims”) of U.S. Patent No. 8,856,505 (“the ’505 Patent”). CrowdStrike respectfully submits that an IPR should be instituted, and that the Challenged Claims should be canceled as unpatentable.

II. REQUIREMENTS FOR IPR—37 C.F.R. §42.104

A. Grounds for Standing Under 37 C.F.R. § 42.104(a)

CrowdStrike certifies that the ’505 Patent is available for IPR. The present petition is being filed within one year of service of a complaint against CrowdStrike. CrowdStrike is not barred or estopped from requesting this review of the Challenged Claims.

B. Challenge and Relief Requested Under 37 C.F.R. §42.104(b)

Petitioner requests IPR of the Challenged Claims on the grounds identified below.

Ground	Claims	Basis for Rejection
1	1-20	Obvious (§103) over Field, Glick, and Shukla

Each reference pre-dates August 7, 2006, which is the priority date of the ’505 Patent (hereinafter “Critical Date”), and qualifies as prior art, as follows:

Reference	Date	Section
Field	Filed 06/30/2006	102(e)
Shukla	Filed 06/27/2007 Priority to 06/29/2006	102(e)
Glick	Filed 02/06/2006	102(e)

III. THE '505 PATENT

A. Brief Description

The '505 Patent describes “a system and method for managing pestware on a protected computer” including “monitoring events during a boot sequence of the computer; managing pestware-related events before native applications can run and after a kernel is loaded; managing pestware-related events when native applications can run; and scanning a registry of the computer for pestware when native applications can run.” EX1001, Abstract. The pestware-related events are monitored by comparing log activity to a set of behavioral rules. *Id.*

With reference to FIG. 3, the '505 Patent describes that previous pestware detection and prevention methods operated as “user-mode services or applications,” which are loaded relatively late in the boot sequence. *Id.*, 1:61-65. This allows pestware to begin execution earlier in the boot sequence and employ “cloaking techniques” to avoid detection by prevention methods that only employ user-mode services or applications. *See id.*, 1:46-67. Thus, the '505 Patent proposes

the implementation of a “kernel level monitor” that “begins execution early in the boot and operating system loading process.” *Id.*, 3:24-31.

However, as will be shown, this method of loading pestware monitoring processes early in the boot sequence was not new. Several prior art references identified the same problem and proposed the same or similar solutions to those claimed the '505 Patent. EX1003, ¶45.

B. Prosecution History

The '505 Patent issued from U.S. Patent App. No. 13/460,655 (the '655 application). In order to gain allowance, the applicant argued that none of the cited prior art taught “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; and scanning data in a registry of the computer for pestware during the second period in the boot sequence.” *Id.*, 147-150. A Notice of Allowance followed and stated that applicant’s arguments were persuasive. *Id.*, 94.

C. Level of Ordinary Skill

A person of ordinary skill in the art relating to the subject matter of the '505 Patent as of August 7, 2006 (“POSITA”) would have possessed a bachelor’s degree in computer science, computer engineering, or an equivalent, as well as two years of industry experience and would have had a working knowledge of endpoint

monitoring systems, software security analysis, and dynamic malware analysis.

EX1003, ¶26. Greater education or greater experience could compensate for a deficiency in either of these criteria. EX1003, ¶26.

D. Claim Construction

All claim terms should be construed according to the *Phillips* standard. *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005); 37 C.F.R. § 42.100. Here, because the Challenged Claims are obvious under any reasonable interpretation, no express constructions are required in this proceeding because “claim terms need only be construed to the extent necessary to resolve the controversy.” *Wellman, Inc. v. Eastman Chem. Co.*, 642 F.3d 1355, 1361 (Fed. Cir. 2011).

To be clear, Petitioner reserves the right to address any construction proposed by Patent Owner or the Board. Petitioner also reserves the right to pursue constructions in district court that are necessary to decide matters of infringement. Furthermore, CrowdStrike is not conceding that each challenged feature satisfies all statutory requirements such as 35 U.S.C. §112. As this is an IPR petition, CrowdStrike is pursuing prior art-based grounds. CrowdStrike is not waiving any arguments concerning other grounds that can only be raised in district court.

IV. THE CHALLENGED CLAIMS ARE UNPATENTABLE

The prior art described below demonstrates both features that the applicant relied on during prosecution.

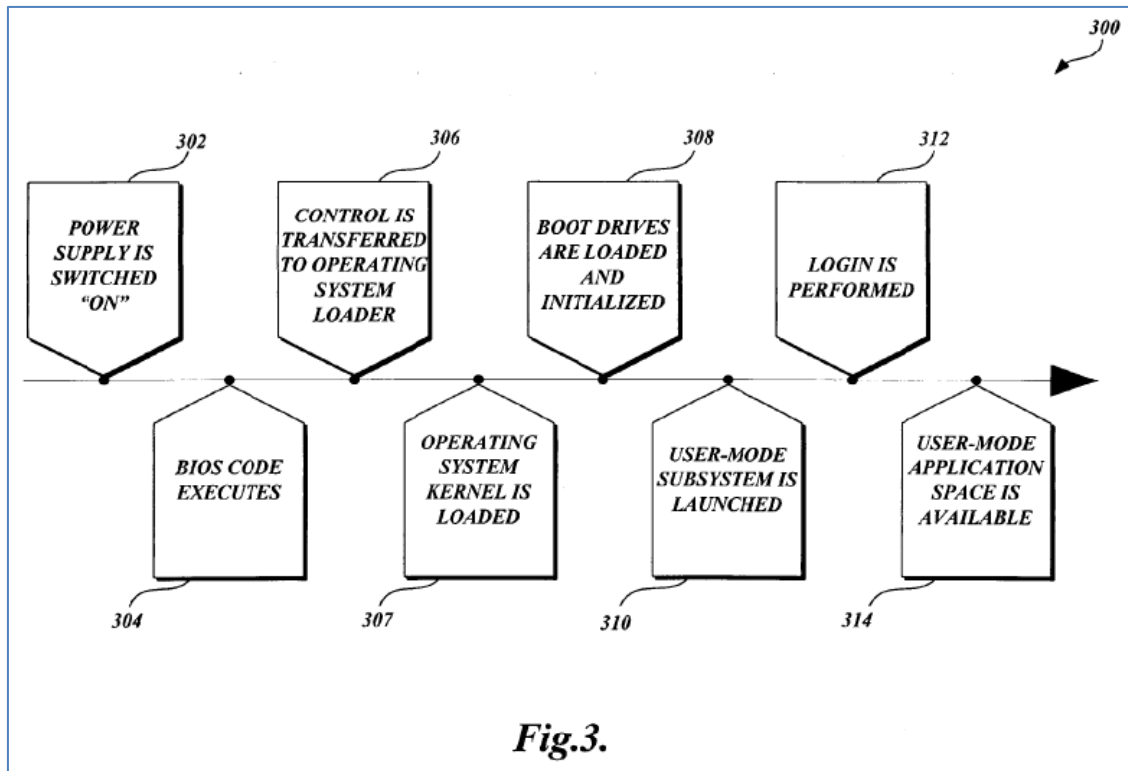
**A. [GROUND 1] – CLAIMS 1-20 WOULD HAVE BEEN OBVIOUS
OVER FIELD, GLICK, AND SHUKLA**

1. Field (EX1004)

Field describes a method “that performs a search for malware during the boot process” and “handles the malware infection.” EX1004, Abstract, 1:7-9. Specifically, the method “causes a software module configured to scan for malware to be initialized at startup. Then, in response to identifying the occurrence of a scanning event, the method causes the software module to search computer memory for data that is characteristic of malware.” *Id.* at [0007]. If the scan does find data characteristic of malware, “the functionality is implemented to prevent malware from executing on the computer.” *Id.*

Like the ’505 Patent, Field recognized that “malware is being distributed with one or more programs specifically designed to ‘hide’ the malware from software designed to protect a computer (e.g., antivirus software, anti-spyware software, and the like).” *Id.*, [0004]. To do so, Field similarly recognized that malware may hook into the operating system and operate “in kernel mode,” and may thus disguise itself from monitoring applications operating in user mode. *See id.*, [0005]. Thus, like the ’505 Patent, Field proposes loading the monitoring program early in the boot sequence to identify and prevent these types of attacks. *See id.*, [0007].

Figure 3 in Field shows a timeline of the events that are performed at computer startup.



Field explains that the “software module may be loaded into memory” at multiple points of the timeline, and may begin executing at events 304, 306, or 310 of the timeline. *Id.* at [0023]. As the captions on Figure 3 show, events 304 and 306 take place before a subsystem of the operating system is loaded (which takes place in event 307). EX1003, ¶55. At or after event 310, the computer is configured to run native applications. *Id.* To identify malware, the scan engine may

search for, for example, “*activities* such as jump instructions in unexpected locations, hidden processes, references to memory addresses that are outside of the range allocated to the operating system, and the like.” EX1004, [0017]².

If malware is identified, “the malware may be removed from the computer or the malware may be ‘quarantined’”. EX1004, [0027]. Quarantining may mean using a “stub” module that is “configured to sustain a malware process, accept and/or return valid data in response to being called, or perform any action to prevent triggering of a malware self-preservation technique.” *Id.* at [0034]. In this way, Field teaches managing or preventing the pestware related event. EX1003, ¶56. Other ways of handling the malware “includes attempting to remove the malware from the computer by killing processes, deleting files, removing entries in configuration files that are associated with the malware, and the like.” *Id.* at [0034].

Field describes that “the scan engine may employ traditional signature-based techniques in a boot environment to detect malware” and that “[o]nce a signature for the malware is available, the scan engine may search data in the memory for a match.” *Id.* at [0017]. In other words, Field suggests the analysis of stored records to identify pestware. However, Field leaves the details of this process to a

² Emphasis added throughout unless noted otherwise.

POSITA by noting that its approach utilizes “traditional . . . techniques.” *Id.* Further, Field notes that “it will be appreciated that various changes can be made” to the operation of its system “without departing from the spirit and scope of the invention.” EX1004, [0036]. From these teachings, a POSITA would have been motivated to turn to known techniques (demonstrated, *e.g.*, by Glick and Shukla) to inform implementation details of Field’s system and method. EX1003, ¶57.

2. Glick (EX1007)

Glick teaches a method “for the detection and removal of threats” which uses a combination of user-mode and kernel-mode detection to identify malware. EX1007 at Title, Abstract; *see also id.*, 3:32-39, 4:11-18. Glick teaches code that “is installed on [a] host computer system and includes [a] user mode application that operates from user mode and [a] kernel mode driver that operates from kernel mode.” *Id.* at 4:11-18. Like Field, Glick describes that the kernel mode portion of its program “is loaded prior to user mode code in the boot process of host computer system 102 to protect privileges being removed by a threat.” *Id.*, 6:23-32; EX1003, ¶58.

Glick provides various implementation details that facilitate communication and coordination between a user mode application and a kernel mode driver to allow for a more capable antivirus/anti-malware protection method. *See* EX1007, 3:21-52; EX1003, ¶59. As part of the capabilities of a kernel mode driver, Glick

recognizes the importance of monitoring the registry, teaching that the “kernel mode driver locates the system privileges in the key descriptor table of the registry and resets the privilege flag to allow [the] user mode application to use the debug privilege of the administrator account.” EX1007, 7:14-19. Moreover, “in one embodiment, [the] kernel mode driver monitors the registry on the host computer system and prevents changes to the registry that prevent a listed file from being deleted on reboot.” *Id.*, 12:37-41.

Furthermore, Glick teaches the importance of reporting significant events. EX1003, ¶60. Specifically, when malicious code is detected, “a record or error message is generated” which, “in one embodiment... identifies the particular malicious code and/or any protective action taken.” EX1007, 8:13-20.

3. Shukla (EXS. 1005, 1006)

Shukla was filed on June 27, 2007, published on Jan 17, 2008, and claims priority to U.S. Provisional Application 60/806,143 (the ’143 provisional application) which was filed on Jun 29, 2006. Shukla qualifies as prior art under 35 U.S.C. §102(e) on the basis of its provisional priority date under *Dynamic Drinkware, LLC, v. National Graphics, Inc.*, 800 F.3d 1375 (Fed. Cir. 2015) and *Amgen v. Sanofi*, 872 F.3d 1367 (Fed. Cir. 2017), which concluded that patents and published patent applications may rely on provisional filing dates as the critical date of

the reference under §102(e) if at least one of the claims of the reference is supported by the provisional application. Here, the '143 provisional application provides a nearly identical specification to Shukla's specification. *Compare* EX1005 to EX1006; EX1003, ¶61. To illustrate, some paragraphs from the "summary of the invention" section of both Shukla and the '143 application are recreated below.

'143 Application, 4

SUMMARY OF THE INVENTION

The present invention provides a new system and method for removing malware and protecting applications against attacks that exploit application vulnerabilities or loopholes to gain unauthorized access into a network or computing device. This approach is significantly different from traditional approaches for intrusion prevention and is "application centric."

In one embodiment of the present invention secures application or any other module in memory with a sandbox that allows that module or application to function properly while preventing attacks from succeeding and causing any harm. For purposes of clarity the term sandbox as used herein means a method to restrict actions taken by an application or a module.

The sandbox for any application is defined by API function calls that determine its interaction with the operating system and are needed by application to function and by other parameters that can restrict any action by the application that can cause irreversible harm or even temporary malfunction. Applications protected by sandbox are able to thwart attacks that exploit vulnerabilities in that application to gain unauthorized access.

The sandbox itself is elastic that is elastic in nature and morphs based on external parameters such as the website or IP address visited by the user, external components installed inside the application, and interaction of the application with other applications on the system.

Shukla, 14-15

SUMMARY OF THE INVENTION

[0011] The present invention provides a new system and method for removing malware and protecting applications against attacks that exploit application vulnerabilities or loopholes to gain unauthorized access into a network or computing device. This approach is significantly different from traditional approaches for intrusion prevention and is "application centric."

[0012] In one embodiment of the present invention secures application or any other module in memory with a sandbox that allows that module or application to function properly while preventing attacks from succeeding and causing any harm. For purposes of clarity the term sandbox as used herein means a method to restrict actions taken by an application or a module.

[0013] The sandbox for any application is defined by API function calls that determine its interaction with the operating system and are needed by application to function and by other parameters that can restrict any action by the application that can cause irreversible harm or even temporary malfunction. Applications protected by sandbox are able to thwart attacks that exploit vulnerabilities in that application to gain unauthorized access.

[0014] The sandbox itself is elastic that is elastic in nature and morphs based on external parameters such as the website or IP address visited by the user, external components installed inside the application, and interaction of the application with other applications on the system.

Because the specifications in Shukla and the '143 application are nearly identical, the disclosure in the '143 application supports at least one of Shukla's claims. In fact, claim 1 of the '143 application (EX1006, p. 31, claim 1) is nearly identical to claim 1 of Shukla (EX1005, pp. 22-23, claim 1), meaning there is literal, word-for-word support for Shukla's claim 1 in the '143 application. EX1003,

¶62. Furthermore, the following table identifies exemplary support in the '143

Provisional application for each limitation of Shukla's independent claim 1:

Claim Limitation (Shukla)	Exemplary support ('143 Provisional)
1. A method for monitoring behavior of plurality of applications or modules in a computing device, comprising the steps of:	Pg 8
injecting a module into the memory space of the said applications;	Pg 9, Figure 4
the injected module monitoring said applications' file system accesses by intercepting API function calls via imported or exported functions table patching and inline hooking of functions at the application layer;	Pg 9, 12
the injected module monitoring said applications' network accesses by intercepting API function calls via imported or exported functions table patching and inline hooking of functions at the application layer;	Pg 9, 12
the injected module monitoring said applications' executable content loading by intercepting API function calls via imported or exported functions table patching and inline hooking of functions at the application layer;	Pg 10-11
the injected module monitoring the memory access by the applications via inline hooks in API function call and the application programming interface functions provided;	Pg 10-11
and the injected module monitoring the registry access by the applications via inline hooks in API function call and the application programming interface functions provided.	Pg 10-11

Shukla teaches a sandbox for detecting, removing, and preventing malware.

EX1005, Title, Abstract; EX1006, 1. The sandbox is constructed "by loading a module or executable code into the application" and, if necessary, "another component may be loaded into the kernel." EX1005, [0035]; EX1006, 9. The sandbox is constructed by hooking into application programming interface (API) function calls

that “can be either user mode or kernel mode functions and the hooking can be done anywhere along the entire chain for the API function call.” EX1005, [0039]; EX1006, 10. Thus, like Field and Glick, Shukla describes the use of both kernel mode and user mode code and techniques to identify and prevent malware. EX1003, ¶63.

Shukla teaches the importance of using behavioral rules, in addition to scanning for signature patterns, describing that “[t]he malware is detected based on its behavior, signature patterns, and code authentication.” EX1005, [0015]; EX1006, 4. “The behavior classification is based on actions taken by applications such as network access, file and registry modifications, API function interception, creation of executable content, stealth behavior, modification of system calls, etc.” EX1005, [0015]; EX1006, 4-5. The behavioral rules can be modified to suit the situation: “[t]he sandbox boundaries themselves are not static and can be tightened or relaxed based on operational parameters...” and “based on which component or module inside the application is executing.” EX1005, [0033]; EX1006, 9. Thus, Shukla provides a more comprehensive method of identifying malware that adds techniques beyond scanning for known signatures. EX1003, ¶64.

Shukla teaches that “[o]nce an application or a module is classified as malicious, its on disk, in registry, and in memory attributes are recorded and used to neutralized [sic] and remove it.” EX1005, [0063]; EX1006, 14.

4. Motivation to combine Field, Glick, and Shukla

In the Field, Glick, and Shukla combination, Glick's user-mode component(s) and record generation and Shukla's behavioral rules and registry scanning are incorporated into Field's method of monitoring and managing malware during the boot sequence. EX1003, ¶66. A POSITA would have been motivated to implement the Field, Glick, and Shukla combination for several reasons.

(i) A POSITA Would Have Found Incorporating Glick's and Shukla's User-Mode Components Into Field Obvious

A POSITA would have been motivated to implement Glick's and Shukla's user-mode/application-level operation into the invention disclosed in Field. EX1003, ¶67. Field teaches that "resources available in a boot environment are limited," and therefore "successfully removing all components of a malware . . . may be difficult or impossible." EX1004, [0034]. Thus, Field teaches that its boot detection module may quarantine the malware, and data about the malware "will typically be communicated to antivirus software that executes once the computer boots so that all of the components of the malware may be removed." *Id.* While Field does not explicitly describe this "antivirus software" as a user-mode application, Glick renders this obvious and provides the implementation details for performing the beneficial functions identified by Field. EX1003, ¶67.

Glick describes that “user mode application 106A includes code that detects malicious code, such as spyware and computer viruses, from user mode and when appropriate takes protective action when malicious code is detected.” EX1007, 3:32-35. Thus, Glick’s user mode application 106A operates like the antivirus software described by Field (i.e., user-mode protection software that loads at the end of the boot sequence and communicates with the kernel-mode software). EX1003, ¶68 (citing EX1004, [0034]). “[W]hen user mode application 106A is unable to perform a selected operation in attempting to detect and/or take protective action, user mode application 106A invokes kernel mode driver 106B assistance.” EX1007, 3:36-39. Thus, based on the combined teachings of Field and Glick, a POSITA would have understood that the kernel-mode component(s) (e.g., the scan engine and boot detection module of Field and the kernel mode driver 106B of Glick) and the user-mode component(s) (e.g., the antivirus software of Field and the user mode application 106A of Glick) are capable of extending each other’s capabilities. EX1003, ¶68. The kernel-mode component(s) are capable of operating and monitoring events during the boot sequence (*see* EX1004, [0022]-[0023]) to prevent a RootKit or other malware from concealing itself (*see* EX1004, [0016]), and also capable of extending the access of the user mode component(s) to certain restricted kernel functionalities (*see* EX1007, 2:30-42, 3:32-52). EX1003, ¶68. Conversely, the user-mode component(s) are capable of extending the access

of the kernel-mode component(s) to certain remediation functions, such as full malware removal (*see* EX1004, [0034]). EX1003, ¶68.

Similar to Glick, Shukla describes a malware detection and removal system that includes both an application-layer component (i.e., the application sandbox) and a “component . . . loaded into the kernel.” EX1005, [0032]-[0036]; EX1006, 8-9. This kernel portion of the malware detection system “hooks into kernel-mode API function calls or even unexported API calls and monitors various components of the kernel.” EX1005, [0041]; EX1006, 10. Shukla’s kernel-level component serves an “important function of being able to detect any rootkit or malware that may compromise the application layer component of the sandbox.” EX1005, [0041]; EX1006, 10-11. Thus, Shukla’s kernel-level component operates in a similar manner to and for a similar purpose of the kernel-level component in each of Field and Glick. EX1003, ¶69.

Shukla describes coordination between its kernel-level and application-layer components to develop and administer flexible behavior rules (described in greater detail below) that allow for better detection of malware. *See* EX1005, [0035], [0039]-[0045], [0056]-[0063]; EX1006, 9, 10-12, 13-15; EX1003, ¶70. Thus, a POSITA would have found it obvious to implement in Field’s system the coordination between the kernel-level and application-layer components described by Shukla, which would allow for the creation of behavior rules that permit Field’s

boot detection module 210 and scan engine 208 to better “search for suspicious activity.” EX1004, [0017]; EX1003, ¶70.

(ii) A POSITA Would Have Found Incorporating Glick’s and Shukla’s Record Generation Into Field Obvious

A POSITA would have been motivated to implement Glick’s record generation in the invention disclosed in Field. EX1003, ¶71. Field implies record generation, though it does not specifically provide the implementation details for doing so. *Id.* For example, as noted above, Field describes communicating data about malware detected during the boot sequence to “antivirus software that executes once the computer boots.” EX1004, [0034]. This implies that a record of signatures is created for pestware-related events. EX1003, ¶71. However, Field does not limit the type or amount of data collected or the manner in which it is collected or communicated. *See id.*; EX1003, ¶71. A POSITA would have been motivated to create a record of the events that the program monitors because it would have allowed for notification of various stakeholders (e.g., a user, system administrator, or security center), which a POSITA would have understood would have permitted better tracking of system security and performance. *See* EX1007, 5:52-67; EX1011, 223-224 (teaching different types of logging used to record information); EX1003, ¶71.

For example, Glick teaches the creation of “a record or error message” that “is generated and provided to the user and/or system administrator” when malicious code is detected. EX1007 at 8:13-20. Glick also teaches an “identifier of the malicious code file” being stored. *Id.* at 13:12-24. As noted above, a POSITA would have recognized that these notifications would have permitted better tracking of system security and performance. EX1003, ¶72.

Similarly, Shukla describes a kernel-component of a malware detection system that hooks into kernel API calls. *See* EX1005, [0043], [0056]-[0057]; EX1006, 11, 13; EX1003, ¶73. Shukla describes a “Learn mode,” in which “kernel functions are hooked and the behavior of the application is monitored.” EX1005, [0110]; EX1006, 24. This “Learn mode” is “used to silently watch the access made by the application without enforcing any rules.” *Id.* Shukla teaches that this monitoring is useful in determining the “risk” of various applications and their behavior, and to develop the behavior rules that are used by the system to detect and remove malware. *See* EX1005, [0110]-[0118]; EX1006, 24-25; EX1003, ¶73. The claims of Shukla refer to these steps as “recording” the events (*e.g.*, “writing executable code to disk or memory, creating startup items, starting new services or drivers or application”) and “generating a report.” *See* EX1005, claim 52; EX1006; claim 52.

From these teachings of Field, Glick, and Shukla, a POSITA would have recognized various benefits (noted above) to configure Field's scan engine 208 and boot detection module 210 to create a record of malware events during the boot process. EX1003, ¶74. For example, because Field's scan engine 208 and boot detection module 210 operate before user-mode/application-level components are loaded and execute, employing record generation during the boot sequence would help combat against rootkits and other threats that attempt to take actions during the boot sequence. *See, e.g.*, EX1004, [0014]; EX1007, 6:27-32; EX1005, [0041], [0072]-[0081]; EX1006, 10-11, 16-19; EX1003, ¶74.

**(iii) A POSITA Would Have Found Incorporating
Shukla's Use of Behavior Rules Into Field Obvious**

A POSITA would have been motivated to implement behavior rules into the invention disclosed in Field because Field already implies the use of behavior rules by stating that "the scan engine may *search for suspicious activities* such as jump instructions in unexpected locations, hidden processes, references to memory addresses that are outside of a range allocated to the operating system, and the like" and that "[a]ntivirus software *may identify activity* that could be characteristic of malware." EX1004, [0017], [0027]. In one example, Field describes that "[r]emoving an entry from a process table or a similar data structure may be *a strong heuristic indicator* that a RootKit is present." EX1004, [0017]. While Field does not specifically describe this functionality as the use of behavior rules to

managing pestware-related events, a POSITA would nonetheless have understood or at least found it obvious that Field's search for suspicious activities would have included the use of behavior rules based on the teachings of Shukla. EX1003, ¶75.

Specifically, Shukla teaches that "a search" can be conducted using "behavior signatures" based on "runtime behavior," such as "network access" or "file creation/deletion." EX1005, [0062]; EX1006, 14. In other words, Shukla describes that the search criteria described in Field may include the search for "behavior signatures" Within runtime data. EX1003, ¶76. Shukla describes that by monitoring "how [an application] interacts with other applications," it is possible to "detect malware content inside an application," which improves upon simple signature-based scanning techniques. EX1005, [0015]-[0016]; EX1006, 4-5. While one of Shukla's embodiments relates to employing these behavior scanning techniques in the context of a sandbox, Field's "invention is also able to remove or neutralized malware on disk and in memory." EX1005, [0017]; EX1006, 5. Thus, when Field describes that *"[a]ny currently existing or yet to be developed techniques may be used* by the scan engine 208 to identify malware that starts executing before services provided by the operating system 204 are available," a POSITA would have understood or at least found it obvious that the use of behavior rules as implied in Field and explicitly described in Shukla was one of these "currently existing . . .

techniques” employed by Field’s scan engine 208. *See* EX1004, [0017]; EX1003, ¶76.

**(iv) A POSITA Would Have Found Incorporating
Shukla’s Registry Scanning Into Field Obvious**

A POSITA would have been motivated to implement registry scanning into the invention disclosed in Field, because Shukla describes that the content of registry entries is a known indication of malware and therefore scanning a registry would have helped Field’s scan engine 208 better identify malware. *See, e.g.*, EX1005, [0008], [0015], [0032], [0062]; EX1006, 2-3, 4-5, 8, 14; EX1003, ¶77. Field already describes the importance of certain data structures maintained by the operating system (e.g., “a process table or a similar data structure”) in identifying the presence of malware such as a RootKit. EX1004, [0017]. For example, to scan the registry, Shukla describes “checking for inconsistencies between the startup list for drivers in the registry compared to the one in the memory.” EX1005, [0084]; EX1006, 20. A POSITA would have recognized or at least found it obvious that one known way to accomplish the comparison described by Shukla would be to scan data in the registry (e.g., the startup list for drivers) and compare the results of that scan to data in the memory that represents what data should be in the registry (sometimes referred to as a clean list). EX1003, ¶77. Thus, a POSITA would have been motivated to implement registry scanning as part of the scan performed by Field’s scan engine 208 to better detect malware capable of hiding itself. *Id.*

5. Element-by-Element Explanation showing the combination of Field, Glick, and Shukla renders challenged claims obvious

(i) Claim 1

[1pre]

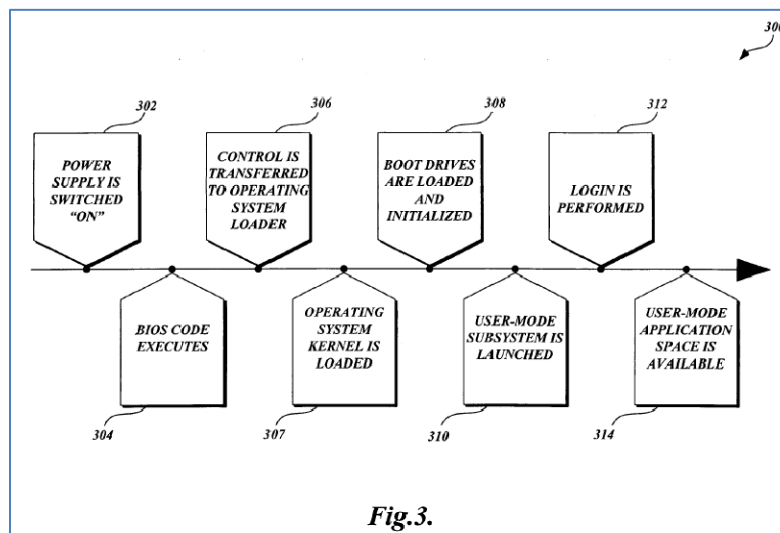
To the extent the preamble is deemed to be limiting, the Field-Glick-Shukla combination renders it obvious. EX1003, ¶¶80-81. The Field-Glick-Shukla combination provides *a method for managing pestware on a computer*. For instance, Field describes a “method [that] causes a software module configured to *scan for malware* to be initialized at *computer* startup...” and “*handle[] the malware* infection” if one is found. EX1004, Abstract. The ’505 Patent describes that “[p]ersonal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as ‘malware’ or ‘pestware.’” EX1001, 1:32-34. Thus, the ’505 Patent uses the terms “malware” and “pestware” interchangeably. EX1003, ¶80.

Similarly, Glick describes an “invention [which] relates to the protection of computer systems from malicious code.” EX1007 at 1:11-12. And Shukla describes a method and apparatus that “detects and removes malware” by “monitoring and controlling the behavior and access privileges of the application and only selectively granting access.” EX1005, Abstract; EX1006, Abstract. Thus, each of Field, Glick, and Shukla—and the combination of the three—describes a “method for managing pestware on a computer,” as recited in [1pre]. EX1003, ¶81.

[1a]

The Field-Glick-Shukla combination describes [1a] or at least renders it obvious.

EX1003, ¶¶82-84. For example, Field, which is titled “identifying malware in a boot environment,” describes “a method... that provides a search for malware *during the boot process.*” EX1004 at Abstract. Figures 3 and 4 of Field also illustrate a malware scan taking place during the boot process, before it is complete. *Id.*, FIGS. 3-4, [0023]-[0026]. “Generally stated, it is desirable to start *scanning for malware as early as possible in the boot process* to prevent malware that infects a previous component from having the opportunity to implement obfuscation techniques.” *Id.*, [0023].



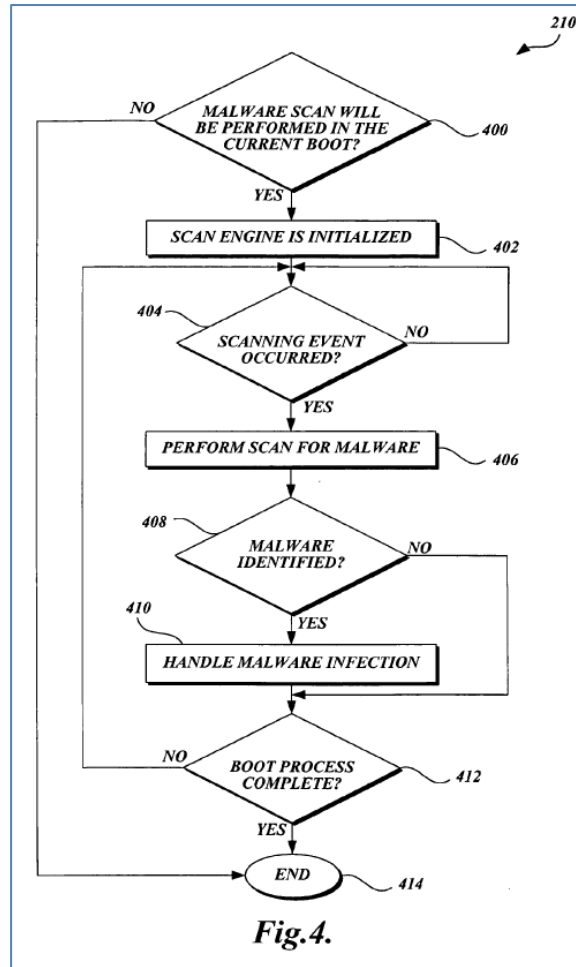


Fig.4.

Field also teaches *monitoring events* during its operations. EX1003, ¶83. Field teaches that, “in response to identifying the occurrence of a scanning event³, the method causes the software module to search computer memory for data that is characteristic of malware.” EX1004, [0007]. For example, Field describes that

³ Field describes a “scanning event” as an event (e.g., the loading of a driver into memory) that triggers a scan for malware. See EX1004, [0031]; EX1003, ¶83.

“the scan engine 208 may *search for suspicious activities* such as jump instructions in unexpected locations, hidden processes, references to memory addresses that are outside of a range allocated to the operating system 204, and the like.” *Id.* at [0017]. A POSITA would have understood or at least found it obvious that these “suspicious activities” described in Field are examples of “events” for which Field’s scan engine 208 monitors. EX1003, ¶83.

Similarly, Shukla describes “a method for *monitoring behavior* of a number of applications or modules in a computing device.” EX1005, [0037]; EX1006, 9. For example, Shukla describes that its system “may monitor the applications file system access or network access by intercepting its API function calls through imported or exported functions table patching and inline hooking of functions at the application layer.” EX1005, [0037]; EX1006, 9-10. In another example, Shukla describes that its system “may monitor the applications executable content, memory access, and registry access in a like manner.” *Id.* Shukla also describes a “Learn mode,” in which “kernel functions are hooked and the behavior of the application is monitored.” EX1005, [0110]; EX1006, 24. And Shukla’s own claims refer to these monitored behaviors as “events.” *See* EX1005, claims 65, 69; EX1006, claims 65, 69. Thus, in the Field-Glick-Shukla combination in which the

scan engine 208 of Field utilizes the behavior rules described by Shukla (*see* Section IV.A.4.iii, *supra*), it would have at least been obvious to monitor events during a boot sequence of the computer, as recited in [1a]. EX1003, ¶84.

[1b]

The Field-Glick-Shukla combination describes [1b] or at least renders it obvious. EX1003, ¶¶85-89. Field teaches *managing pestware-related events*. For example, Field describes identifying “activity that could be characteristic of malware” and removing or quarantining identified malware “from the computer”. EX1004, [0016], [0027]. In one specific example implementation, Field describes a computer 200 that includes a scan engine 208 and a boot detection module 210, which “collectively provide *a way to identify malware*, such as a RootKit, that starts executing before the services provided by traditional antivirus software are available.” EX1004, [0014]. As described with respect to [1a], Field describes that “the scan engine 208 may *search for suspicious activities* such as jump instructions in unexpected locations, hidden processes, references to memory addresses that are outside of a range allocated to the operating system 204, and the like.” *Id.*, [0017].

If such malware is identified, “the boot detection module 210 causes the malware infection to be handled,” which may include, for example, “attempting to

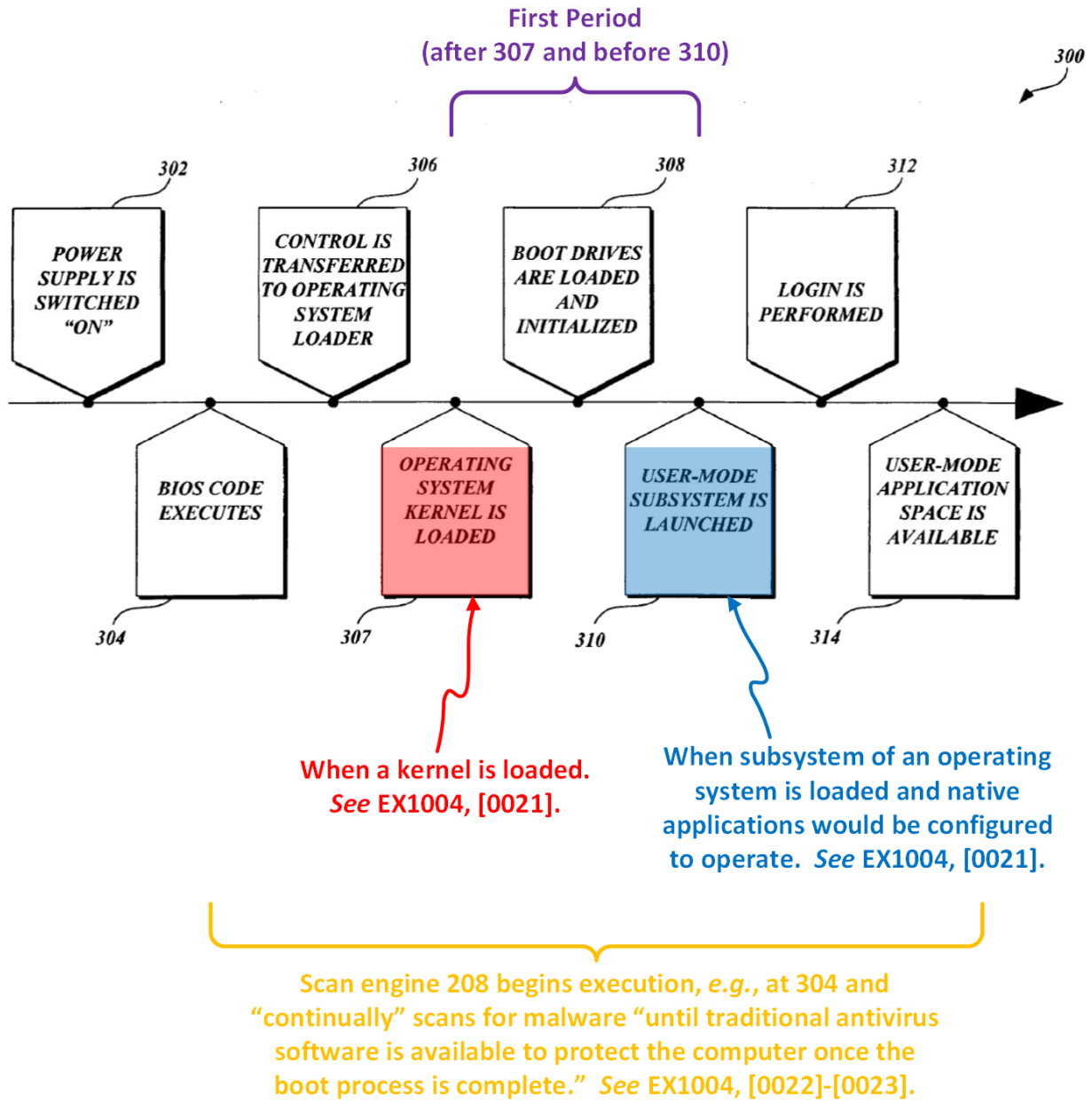
remove the malware from the computer by killing processes, deleting files, removing entries in configuration files that are associated with the malware, and the like.” *Id.*, [0034].

A POSITA would have recognized or at least found it obvious that the search for malware and handling of the malware identified in the search is a process of “managing pestware-related events,” as recited in [1b]. EX1003, ¶86. For example, a POSITA would have recognized or at least found it obvious that identifying the removal of “an entry from a process table or a similar data structure” as a pestware-related event, and that then removing the associated malware to be at least one way to “manage” that event. *Id.*

Moreover, for the reasons described in Section IV.A.4.iii, *supra*, a POSITA would have found it obvious to utilize the behavior rules described by Shukla in Field’s scan engine 208. Shukla describes “a method for ***monitoring behavior*** of a number of applications or modules in a computing device.” EX1005, [0037]; EX1006, 9. For example, Shukla describes that its system “may monitor the applications file system access or network access by intercepting its API function calls through imported or exported functions table patching and inline hooking of functions at the application layer.” EX1005, [0037]; EX1006, 9-10. In another example, Shukla describes that its system “may monitor the applications executable content, memory access, and registry access in a like manner.” *Id.* Shukla’s own

claims refer to these monitored behaviors as “events.” *See* EX1005, EX1006, claims 65, 69. Thus, in the Field-Glick-Shukla in which the scan engine 208 of Field utilizes the behavior rules described by Shukla (*see* Section IV.A.4.iii, *supra*), it would have at least been obvious to ***manage pestware-related events*** by utilizing the behavior rules of Shukla to identify the pestware-related events, and then “handle” the malware associated with those events using Field’s boot detection module 210. EX1003, ¶87.

Field teaches managing pestware-related events during the claimed ***first period***. For example, Field describes that “a boot detection module 210 is provided that causes malware to be identified ***before the services provided by an operating system are available***.” EX1004, [0016]. Generally, “the boot detection module 210 causes the scan engine 208 to be loaded into memory and executed ***during the boot process***.” *Id.* “More specifically, at any number of different locations in the timeline 300, a software module (e.g., the scan engine 208) may be loaded into memory. EX1004, [0022]. An annotated version of timeline 300 in FIG. 3 is provided below. EX1003, ¶88.



Generally, Field discloses that it "is desirable to start scanning for malware as early as possible in the boot process to prevent malware that infects a previous component from having the opportunity to implement obfuscation technique." EX1004, [0023]. More specifically, Field's specification states that its scan engine

“may be loaded into memory” at “any number of different locations in the time-line” and may “begin executing at events **304, 306**, or 310.” *Id.*, [0022]-[0023].

“In this regard, components that execute during the boot process may be continually scanned until traditional antivirus software is available to protect the computer once the boot process is complete.” *Id.*, [0022]. Thus, as shown in the above annotation of FIG. 3, where a POSITA configured the scan engine 208 to begin executing at events 304 or 306 per the explicit teachings of Field, the scan engine 208 would be executing (e.g., managing pestware-related events by scanning for malware and removing or quarantining identified malware) during a **first period** that is **before the computer becomes configured to run native applications, before a subsystem of an operating system is loaded (event 310)**, and **after a kernel is loaded (event 307)**. EX1003, ¶89.

[1c]

The Field-Glick-Shukla combination describes [1b] or at least renders it obvious. EX1003, ¶¶90-96. Field teaches *managing pestware-related events*. For example, Field describes identifying “activity that could be characteristic of malware” and removing or quarantining identified malware “from the computer”. EX1004, [0016], [0027]. In one specific example implementation, Field describes a computer 200 that includes a scan engine 208 and a boot detection module 210, which “collectively provide *a way to identify malware*, such as a RootKit, that

starts executing before the services provided by traditional antivirus software are available.” EX1004, [0014]. As described with respect to [1a], Field describes that “the scan engine 208 may *search for suspicious activities* such as jump instructions in unexpected locations, hidden processes, references to memory addresses that are outside of a range allocated to the operating system 204, and the like.” *Id.*, [0017].

If such malware is identified, “the boot detection module 210 causes the malware infection to be handled,” which may include, for example, “attempting to remove the malware from the computer by killing processes, deleting files, removing entries in configuration files that are associated with the malware, and the like.” *Id.*, [0034].

A POSITA would have recognized or at least found it obvious that the search for malware and handling of the malware identified in the search is a process of “managing pestware-related events,” as recited in [1b]. EX1003, ¶91. For example, a POSITA would have recognized or at least found it obvious that identifying the removal of “an entry from a process table or a similar data structure” as a pestware-related event, and that then removing the associated malware to be at least one way to “manage” that event. *Id.*

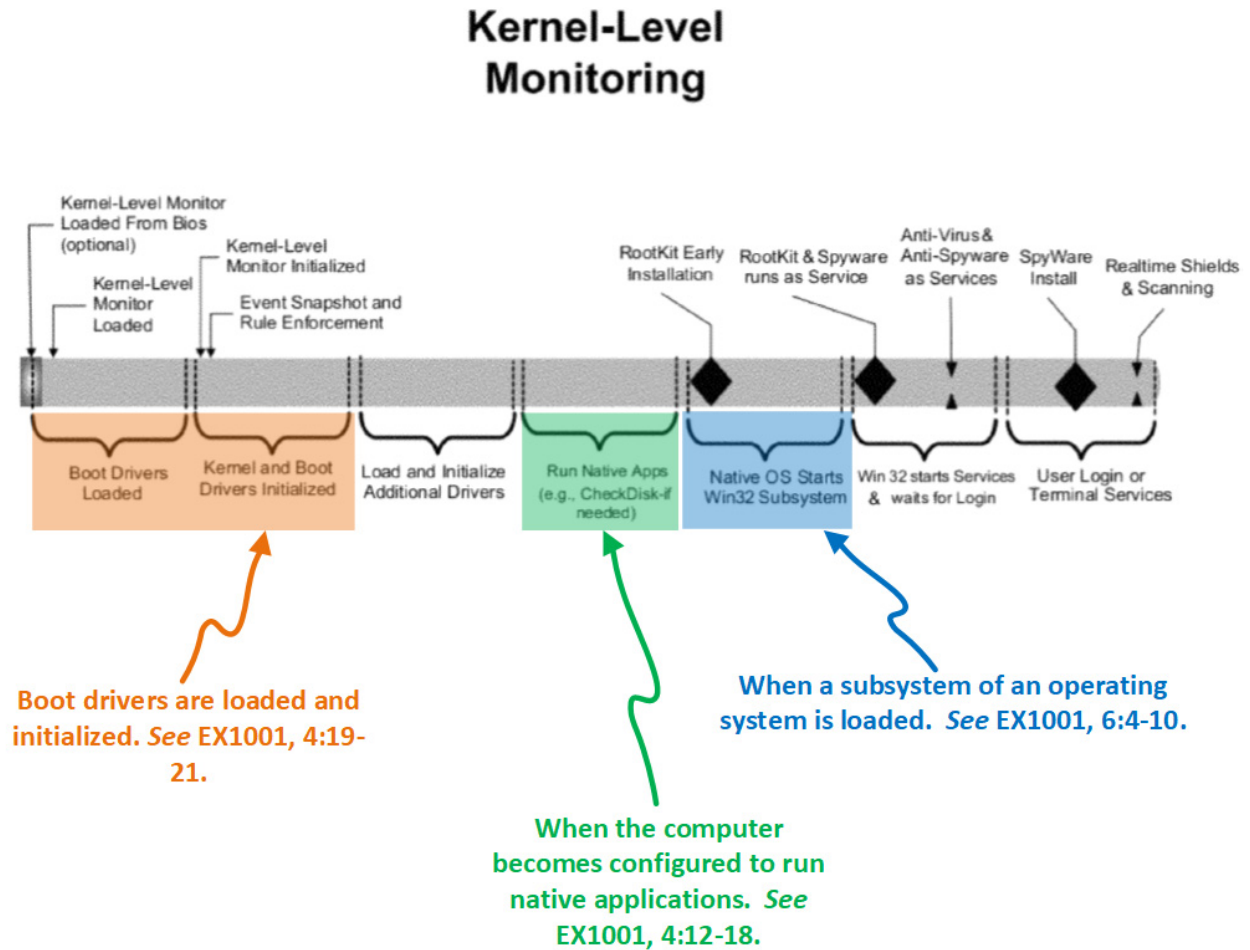
Moreover, for the reasons described in Section IV.A.4.iii, *supra*, a POSITA would have found it obvious to utilize the behavior rules described by Shukla in

Field's scan engine 208. Shukla describes "a method for ***monitoring behavior*** of a number of applications or modules in a computing device." EX1005, [0037]; EX1006, 9. For example, Shukla describes that its system "may monitor the applications file system access or network access by intercepting its API function calls through imported or exported functions table patching and inline hooking of functions at the application layer." EX1005, [0037]; EX1006, 9-10. In another example, Shukla describes that its system "may monitor the applications executable content, memory access, and registry access in a like manner." *Id.* Shukla's own claims refer to these monitored behaviors as "events." *See* EX1005, EX1006, claims 65, 69. Thus, in the Field-Glick-Shukla in which the scan engine 208 of Field utilizes the behavior rules described by Shukla (*see* Section IV.A.4.iii, *supra*), it would have at least been obvious to ***managing pestware-related events in accordance with a set of behavior rules*** by utilizing the behavior rules of Shukla to identify the pestware-related events, and then "handle" the malware associated with those events using Field's boot detection module 210. EX1003, ¶92.

With respect to the claimed ***second period***, the '505 Patent describes that "a boot sequence of the protected computer is initiated (e.g., by a user of the protected computer) (Block 202), and in advance of ***a period when native applications (e.g., checkdisk)*** of the protected computer beginning execution, the kernel-level monitor 105 is initialized." EX1001, 4:12-18. Thus, FIG. 4 of the '505 patent shows

that an operating system is able to run native applications after drivers are loaded and before the native operating system starts its subsystem, annotated below.

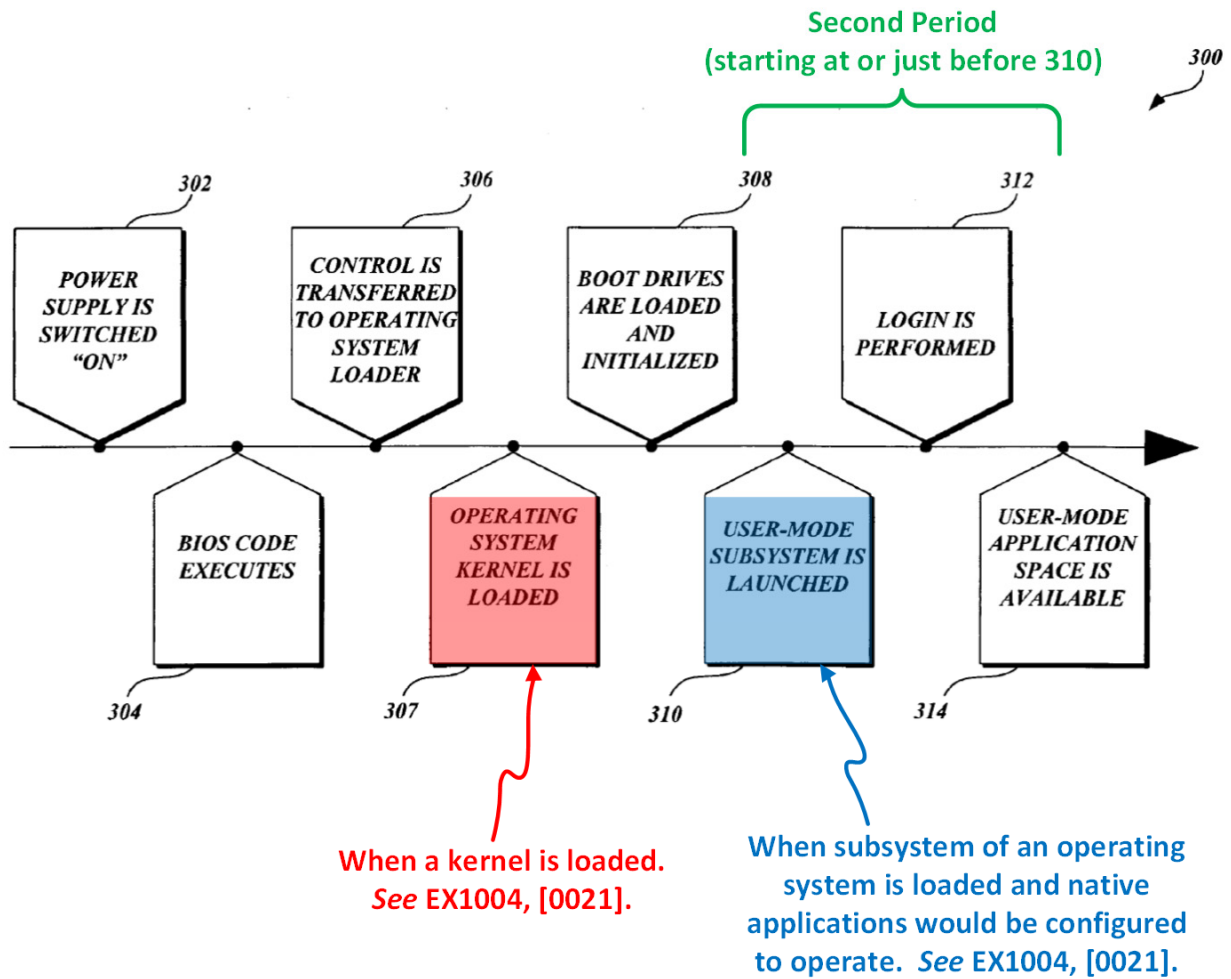
EX1001, FIG. 4; EX1003, ¶93.



A POSITA would have understood or at least found it obvious that Field's computer becomes configured to run native applications around **event 310**.

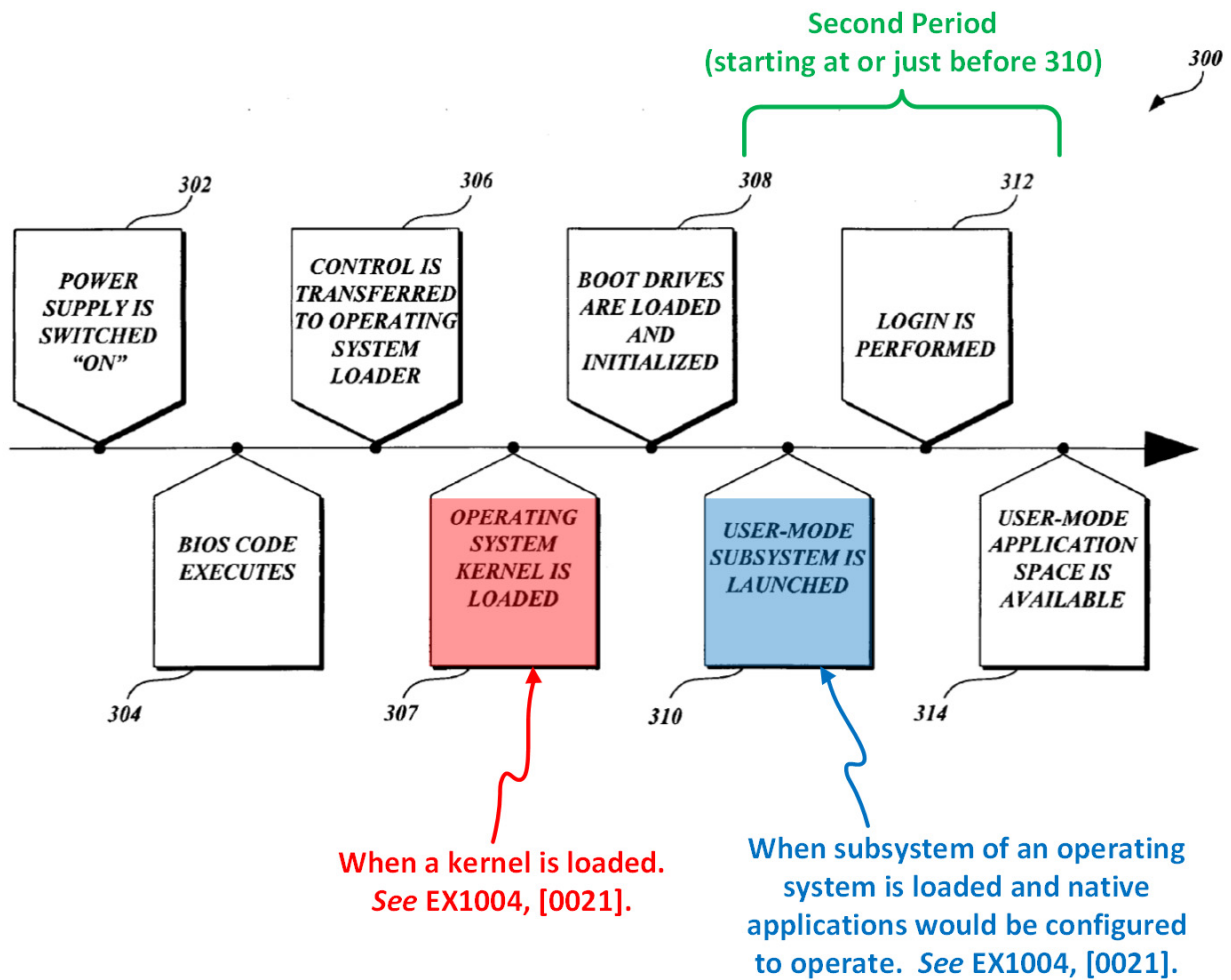
EX1003, ¶94. For example, Field describes that “the user mode subsystem provides support services to the *user mode application space*.” EX1004, [0021].

“Once the user mode services are available, at event 314, the user-mode application space may be accessed to execute programs. Once event 314 is reached, programs designed to perform specific tasks on a general purpose computer may be executed.” *Id.*



A POSITA would have understood or at least found it obvious that “native applications” would have been configured to run at approximately the same time and certainly no later than when the “user-mode applications” described by Field would have been configured to run. EX1003, ¶95. The timeline in FIG. 4 of the ’505 Patent describes effectively the same sequence of events in a standard boot process and the timeline in FIG. 3 of Field. *Id.* The methods and systems in each of the ’505 Patent and Field are described as being operable on general purpose computers that run a variety of operating systems, including Microsoft Windows. EX1001, 3:5-8 (“And if implemented in software, the modules can be designed to operate on any type of computer system including WINDOWS and Linux-based systems”); EX004, [0015] (“The operating system 204 illustrated in FIG.2 may be a general-purpose operating system such as a Microsoft® operating system, UNIX® operating system, or Linux® operating system.”); EX1003, ¶95. Thus, while the exact events indicated on each of the timelines are not identical, a POSITA would nevertheless have understood the two timelines as depicting the same general boot process flow, which would have been specific to a given operating system and computer configuration and independent of the monitoring system/method. EX1003, ¶95. Accordingly, because the ’505 Patent describes that **the computer is configured to run native applications** between when **the boot**

drivers are loaded and initialized and when a subsystem of an operating system is loaded, for the same reasons, a POSITA would have understood or at least found it obvious that Field's computer is configured to run native applications around or just before Field's event 310 where Field describes that the "user mode subsystem is launched." EX1004, [0021]; EX1003, ¶95.



Generally, Field discloses that “is desirable to start scanning for malware as early as possible in the boot process to prevent malware that infects a previous component from having the opportunity to implement obfuscation technique.” EX1004, [0023]. More specifically, Field’s specification states that its scan engine “may be loaded into memory” at “any number of different locations in the timeline” and may “begin executing at events **304, 306**, or 310.” *Id.*, [0022]-[0023]. “In this regard, components that execute during the boot process may be continually scanned until traditional antivirus software is available to protect the computer once the boot process is complete.” *Id.*, [0022]. Thus, as shown in the above annotation of FIG. 3, where a POSITA configured the scan engine 208 to begin executing at events 304 or 306 per the explicit teachings of Field, the scan engine 208 would be executing (e.g., managing pestware-related events by scanning for malware and removing or quarantining identified malware) during **a second period when the computer is configured to run native applications**. EX1003, ¶96.

[1d]

The Field-Glick-Shukla combination describes [1d] or at least renders it obvious. EX1003, ¶¶97-101. As described with respect to [1a]-[1c], Field generally describes that the boot detection module 210 and scan engine 208 “search[es] for suspicious activity” and, if malware is detected, “causes the malware infection to

be handled.” *See* EX1004, [0017], [0027], [0032], [0034]. When the boot detection module 210 and scan engine 208 detect potential malware, Field describes that it “may be difficult or impossible” to “successfully remov[e] all components of a malware” using the limited “resources available in a boot environment.” EX1004, [0034]. Thus, Field describes that, “when the malware is quarantined, data will typically be communicated to antivirus software that executes once the computer boots so that all of the components of the malware may be removed.” EX1004, [0034]. Thus, Field suggests that certain records are generated and stored by the boot detection module 210 and scan engine 208 (i.e., the kernel-mode components) and communicated to antivirus software (e.g., user-mode components) so that the detected malware can be further handled after the boot process is complete. *See* EX1004, [0034]; EX1003, ¶97. Field does not prescribe or otherwise limit the form of the records about the potential malware that are generated and stored during the boot process that is passed to the post-boot antivirus software. *Id.*

Based on the teachings of Glick and Shukla, as well as the general knowledge of a POSITA, it would have been obvious to a POSITA that the records generated by Field’s boot detection module 210 and scan engine 208 and passed to the post-boot antivirus software would include a record of events that includes pestware-related events. EX1003, ¶98. A POSITA at the time of filing would have been motivated to create a record of the events that the program monitors. *Id.*

For example, Glick describes that, any time malicious code is detected, “*a record* or error message *is generated* and provided to the user and/or system administrator,” and the record may include information that identifies “the particular malicious code and/or any protective action taken.” *See* EX1007, 5:52-59. These records can be written to a file or otherwise logged. *See* EX1007, 5:60-64; EX1003, ¶98.

Similarly, as explained in Section IV.A.4.i, *supra*, Shukla describes malware detection and removal system that includes both an application-layer component (i.e., the application sandbox) and a “component . . . loaded into the kernel.” EX1005, [0032]-[0036]; EX1006, 8-9. This kernel portion of the malware detection system “hooks into kernel-mode API function calls or even unexported API calls and monitors various components of the kernel.” EX1005, [0041]; EX1006, 10. Shukla’s kernel-level component serves an “important function of being able to detect any rootkit or malware that may compromise the application layer component of the sandbox.” EX1005, [0041]; EX1006, 10-11. Thus, Shukla’s kernel-level component operates in a similar manner to and for a similar purpose of the kernel-level component in each of Field and Glick. EX1003, ¶99.

Shukla describes that its kernel-level component works by hooking in various API calls within the kernel that allows Shukla’s malware detection system to observe and control the execution of these API calls. *See* EX1005, [0043], [0056] ;

EX1006, 11, 13; EX1003, ¶100. “Target functions are primarily related to file manipulation, process creations, network access, registry access, memory access etc.” EX1005, [0056]; EX1006, 13. “After the target functions are hooked in kernel, the processing of intercepted API function calls may be done in conjunction with the application sandbox *or independently*.” EX1005, [0057]; EX1006, 13. In other words, Shukla describes that its kernel-level component can work in conjunction with an application-layer component or independently of one. *Id.*; EX1003, ¶100. A POSITA would have recognized that this ability to operate independently makes the methods employed by Shukla’s kernel-level component ideal for implementing the “suspicious activity” monitoring performed by Field’s scan engine 208, which operates prior to the loading and initialization of an application-level component with which it later communicates and coordinates. *Id.*; EX1004, [0017] (“Any currently existing or yet to be developed techniques may be used by the scan engine 208 to identify malware Moreover, the scan engine 208 may search for suspicious activities such as jump instructions in unexpected locations, hidden processes, references to memory addresses that are outside of a range allocated to the operating system 204, and the like.”), [0034] (“when the malware is quarantined, data will typically be communicated to antivirus Software that executes once the computer boots so that all of the components of the malware may be removed.”); EX1003, ¶100.

In order to create the rules upon which Shukla's malware detection system evaluates behavior, Shukla describes a "Learn mode," in which "kernel functions are hooked and the behavior of the application is monitored." EX1005, [0110]; EX1006, 24; EX1003, ¶101. This "Learn mode" is "used to silently watch the access made by the application without enforcing any rules." *Id.* Shukla teaches that this monitoring is useful in determining the "risk" of various applications and their behavior, and to develop the behavior rules that used by the system to detect and remove malware. *See* EX1005, [0110]-[0118]; EX1006, 24-25; EX1003, ¶101. From these teachings, a POSITA would have been motivated to configure Field's scan engine 208 and detection module 210 to hook into certain kernel APIs (e.g., those "related to file manipulation, process creations, network access, registry access, memory access etc.," as taught by Shukla) and record the behavior of the drivers and applications that are loaded and begin execution during the boot cycle. *See* EX1005, [0110]-[0118]; EX1006, 24-25; EX1003, ¶101. Through the creation of these records, the application-level component of the combined system would advantageously be able to statistically analyze the observed and recorded behavior from the boot process and better protect the system from threats that begin during the boot process (e.g., rootkits). *Id.* Configured in this manner, the Field-Glick-Shukla combination would "generat[e], in response to the monitoring, a record of

events, the record of events including the pestware-related events,” as recited in [1d]. EX1003, ¶101.

[1e]

The Field-Glick-Shukla combination describes [1e] or at least renders it obvious. EX1003, ¶¶102-103. As described above with respect to [1d], Field describes that, “when the malware is quarantined, data will typically be communicated to antivirus software that executes once the computer boots so that all of the components of the malware may be removed.” EX1004, [0034]. Thus, Field suggests that certain records are generated and stored by the boot detection module 210 and scan engine 208 (i.e., the kernel-mode components) and communicated to antivirus software (e.g., user-mode components) so that the detected malware can be further handled after the boot process is complete. *See* EX1004, [0034]; EX1003, ¶102.

As further described above with respect to [1d], Shukla describes a “Learn mode,” in which “kernel functions are hooked and the behavior of the application is monitored.” EX1005, [0110]; EX1006, 24. This “Learn mode” is “used to silently watch the access made by the application without enforcing any rules.”⁴ *Id.*

⁴ A POSITA would have understood or at least found it obvious that the system of Shukla would still be able to enforce rules for those applications for which rules

Shukla teaches that this monitoring is useful in determining the “risk” of various applications and their behavior, and to develop the behavior rules that used by the system to detect and remove malware. *See* EX1005, [0110]-[0118]; EX1006, 24-25; EX1003, ¶103. From these teachings a POSITA would have found it obvious to configure the application-level component of the Field-Glick-Shukla combination to “analyz[e] the record of events so as to identify the pestware-related events” (as recited in [1e]), because this analysis would allow the application-level component of the combined system to statistically assess the risk factor of the behaviors observed during the boot cycle and develop rules to identify and remove malware. *See id.*

[1f]

The Field-Glick-Shukla combination describes [1f] or at least renders it obvious. EX1003, ¶¶104-105. As described above with respect to [1d] and [1e], as well as in Section IV.A.4.iii, *supra*, a POSITA would have been motivated to configure Field’s scan engine 208 and detection module 210 to hook into certain ker-

have already been created while in the described “learn mode,” as this would have avoided a system in which the “learn mode” leaves the system vulnerable.

EX1003, ¶103.

nel APIs (e.g., those “related to file manipulation, process creations, network access, registry access, memory access etc.,” as taught by Shukla) and record the behavior of the drivers and applications that are loaded and begin execution during the boot cycle. *See* EX1005, [0110]-[0118]; EX1006, 24-25; EX1003, ¶104.

Shukla teaches that this “Learn mode” process is used to generate and test behavior rules that will be used to identify malware, and that the process ensures proper functioning and security. *See* EX1005, [0110]; EX1006, 24; EX1003, ¶104.

A POSITA would have understood that iteratively observing new behavior and assessing it would have allowed the system of the Field-Glick-Shukla combination to better adapt to new threats by allowing for the creation of new behavior rules and the modification of old behavior rules. *See* EX1005, [0078]; EX1006, 18; EX1003, ¶105. As Shukla recognized, using rigid signature-based detection techniques was “ineffective” (*see* EX1005, [0078]; EX1006, 18) and that using rigid behavior rules leads to a malware detection system that is either overly burdensome on the user or overly permissive to malware (*see* EX1005, [0119]-[0123]; EX1006, 25-26). EX1003, ¶105. Thus, Shukla teaches the benefits of flexibility in the creation, maintenance, and application of behavior rules to adapt to new malware threats. From these teachings, a POSITA would have found it obvious for the Field-Glick-Shukla combination to “modify[] the set of behavior rules so as to prevent the pestware related events,” as recited in 1[f]. EX1003, ¶105.

1[g]

The Field-Glick-Shukla combination describes [1g] or at least renders it obvious. EX1003, ¶¶106-107. During the original prosecution of the '505 Patent, Applicant distinguished a prior art reference by arguing that “[d]etecting suspicious reads of a registry is not the same as scanning data in a registry of the computer for pestware during the second period in the boot sequence.” EX1002, 151 (emphasis in original). Thus, under the Applicant’s own interpretation, the claims require “scanning data in a registry of the computer,” and not just monitoring reads of data. *Id.* The Field-Glick-Shukla combination renders this limitation, under Applicant’s own interpretation, obvious. EX1003, ¶106.

As described in Section IV.A.4.iv, *supra*, Field describes the importance of certain data structures maintained by the operating system (e.g., “a process table or a similar data structure”) in identifying the presence of malware such as a RootKit. EX1004, [0017]. A POSITA would have understood or at least found it obvious based on the teachings of Shukla that the registry is one example of an operating system data structure, and that scanning the registry “can either reveal malicious intent or hint at potentially malicious nature.” EX1005, [0062], [0068]; EX1006, 14, 15; EX1003, ¶107. For example, to scan the registry, Shukla describes “checking for inconsistencies between the startup list for drivers in the registry compared to the one in the memory.” EX1005, [0084]; EX1006, 20. A POSITA would have

recognized or at least found it obvious that one known way to accomplish the comparison described by Shukla would be to scan data in the registry (e.g., the startup list for drivers) and compare the results of that scan to data in the memory that represents what data should be in the registry (sometimes referred to as a clean list). EX1003, ¶107. Indeed, such registry scanning was well known to POSITAs as a standard part of monitoring pestware-related events. *See e.g.* EX1010, 3:35-37 (“System configuration files such as *a system registry are scanned for specific entries* that are associated with active malware.”); EX1003, ¶107. From these teachings, a POSITA would have found it obvious for the Field-Glick-Shukla combination to **scan[] data in a registry of the computer for pestware during the second period in the boot sequence.** EX1003, ¶107.

(ii) **Claim 2**

2[a]

As described above in Section IV.A.4.i, *supra*, a POSITA would have found it obvious to implement Glick’s and Shukla’s user-mode/application-level operation into the invention disclosed in Field. EX1003, ¶109. Field teaches that “resources available in a boot environment are limited,” and therefore “successfully removing all components of a malware . . . may be difficult or impossible.” EX1004, [0034]. Thus, Field teaches that its boot detection module may quarantine the malware, and data about the malware “will typically be communicated to

antivirus software that executes once the computer boots so that all of the components of the malware may be removed.” *Id.*

Field teaches that “traditional antivirus software” (i.e., a user-mode/application-level component) is not available until after the boot sequence. *See* EX1004, [0014], [0022] (“components that execute during the boot process may be continually scanned until traditional antivirus software is available to protect the computer once the boot process is complete.”); EX1003, ¶110. Indeed, this is consistent with the ’505 Patent’s own description of the prior art with respect to FIG. 3, which shows that “user-mode services or applications” such as “anti-virus” and “anti-spyware” services are initiated after the boot and operating system sequences. EX1001, FIG. 3, 1:61-65; EX1003, ¶110.

Glick and Shukla both describe implementations details for a user-mode/application-level component that a POSITA would have found it obvious to utilize in Field’s malware detection and removal system as part of or in place of the “traditional antivirus software” that Field relies upon to fully remove malware identified during the boot sequence. *See* EX1004, [0034]; Section IV.A.4.i, *supra*. Based on the teachings of Field and Glick, a POSITA would have recognized or at least found it obvious that these user-mode/application-level components would have been launched at the end of or after the boot sequence, and thus **after an operating system is initiated**. *See* EX1004, [0022]; EX1007, 6:23-32; EX1003, ¶111.

Based on the teachings of Field, Glick, and Shukla, a POSITA would have further recognized or found it obvious that these user-mode/application-level components together constitute a **pestware management engine**. EX1003, ¶112. As noted above, Field teaches that its boot detection module may quarantine the malware, and data about the malware “will typically be communicated to antivirus software that executes once the computer boots *so that all of the components of the malware may be removed.*” EX1004, [0034]. A POSITA would have recognized or at least found it obvious that the removal of malware is an integral part of pestware management. EX1003, ¶112. Additionally, the implementation details of the user-mode/application-level components provided by Glick and Shukla confirm that these components constitute a pestware management engine. EX1003, ¶112. For example, Glick describes that “user mode application 106A includes code that detects malicious code, such as spyware and computer viruses, from user mode and when appropriate takes protective action when malicious code is detected.” EX1007, 3:32-39. Similarly, Shukla describes that its system includes both application-level and kernel-level components, and that the application-level component “prevents attacks, detects and removes malware based on an elastic or adaptive limits on application behavior” by “blocking, permitting, or monitoring the API function calls at the application layer.” See EX1005, [0032]-[0035],

[0040]-[0041]; EX1006, 8-9, 10-11; EX1003, ¶112. These example functions described by Glick and Shukla are consistent with the functions that the pestware management engine of the '505 Patent performs. *See, e.g.*, EX1001, 5:34-59; EX1003, ¶112. Based at least on these teachings, a POSITA would have further recognized or found it obvious that these user-mode/application-level components together constitute a **pestware management engine**. EX1003, ¶112.

2[b]

As described in Section IV.A.4.iii and with respect to [1d]-[1f], *supra*, a POSITA would have found it obvious and been motivated to implement the creation, modification, and enforcement of behavior rules into the invention disclosed in Field, as described by Shukla. EX1003, ¶113.

Shukla describes a “Learn mode,” in which “kernel functions are hooked and the behavior of the application is monitored.” EX1005, [0110]; EX1006, 24. This “Learn mode” is “used to silently watch the access made by the application without enforcing any rules.” *Id.* Shukla teaches that this monitoring is useful in determining the “risk” of various applications and their behavior, and to develop the behavior rules that used by the system to detect and remove malware. *See* EX1005, [0110]-[0118]; EX1006, 24-25; EX1003, ¶114. Because Shukla teaches that the “Learn mode” is based on hooking of both application and kernel func-

tions, a POSITA would have understood or at least found it obvious that rule creation and compilation process is based on coordination between both the kernel-level and application-level components of the system. *See* EX1005, [0110]-[0118]; EX1006, 24-25. Given that the application-level components of these types of malware detection systems were known to have broader processing and remediation capabilities (*see, e.g.*, EX1004, [0034]), a POSITA would have understood or at least found it obvious that the user-mode/application-level components of the Field-Glick-Shukla combination would have been configured to compile the behavior rules. EX1003, ¶114. Indeed, this is consistent with Glick’s general descriptions of its user mode application 106A performing the primary analysis and coordination functions, and calling upon the kernel mode driver 106B for specific functional assistance. *See, e.g.*, EX1007, 12:50-13:44, 18:6-33; EX1003, ¶114. Thus, in the Field-Glick-Shukla combination, a POSITA would have understood or at least found it obvious that “the set of behavior rules includes behavior rules compiled by the pestware management engine,” as recited in 2[b]. EX1003, ¶114.

(iii) Claim 3

[3]

As described above with respect to Section IV.A.4.ii and [1d], *supra*, Shukla describes that its kernel-level component works by hooking in various API calls within the kernel that allows Shukla’s malware detection system to *observe* and

control the execution of these API calls. *See* EX1005, [0043], [0056]; EX1006, 11, 13; EX1003, ¶116. Shukla describes a “Learn mode,” in which “kernel functions are hooked and the behavior of the application is monitored.” EX1005, [0110]; EX1006, 24. This “Learn mode” is “used to silently watch the access made by the application without enforcing any rules.” *Id.* The claims of Shukla refer to these steps as “recording” the events (*e.g.*, events such as “writing executable code to disk or memory, creating startup items, starting new services or drivers or application”) and “generating a report.” *See* EX1005, claim 52; EX1006, claim 52.

According to Shukla, the “[t]arget functions” that the kernel-level component hooks “are primarily related to file manipulation, process creations, network access, registry access, memory access etc.” EX1005, [0056]; EX1006, 13; EX1003, ¶117. “Once an application or a module is classified as malicious, *its // disk, in registry, and in memory attributes are recorded* and used to neutralized and remove it.” EX1005, [0063]; EX1006, 14-15. Similarly, Shukla describes that “[o]nce a hooked module is discovered, *it is flagged* and its execution path is traced to discover any other modules that may be hooked into that function.” EX1005, [0073]-[0074]; EX1006, 16-17.

From these teachings, a POSITA have understood or at least found it obvious that **the record of events includes information selected from a group**

consisting of: process identification information, file identification

information, and hook generation information, as these various information are consistent with the information described as being record in Shukla (e.g., the “disk, in registry, and in memory attributes”) and would have allowed the Field-Glick-Shukla combination “to neutralize[] and remove” the malware that is detected, as is taught by Field and Shukla. *See* EX1004, [0022], [0034]; EX1005, [0043], [0056], [0110], claim 52; EX1006, 11, 13, 24, claim 52; EX1003, ¶118.

(iv) Claim 4

[4]

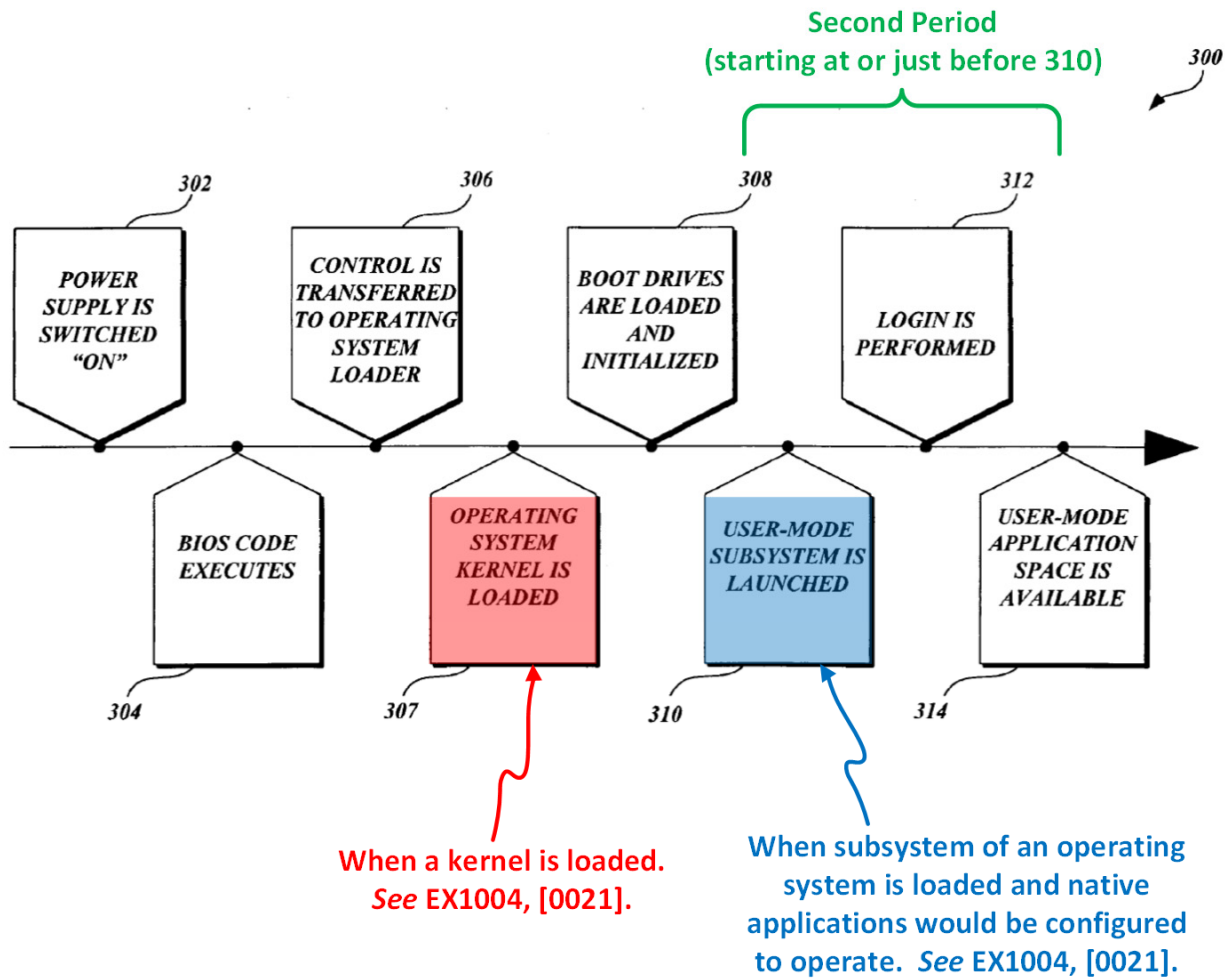
As described above with respect to [1c], FIG. 4 of the '505 patent shows that an operating system is able to run native applications after drivers are loaded and before the native operating system starts its subsystem. EX1003, ¶120.

A POSITA would have understood or at least found it obvious that Field's computer becomes configured to run native applications around event 310.

EX1003, ¶121. For example, Field describes that “the user mode subsystem provides support services to the *user mode application space*.” EX1004, [0021].

“Once the user mode services are available, at event 314, the user-mode application space may be accessed to execute programs. Once event 314 is reached, programs designed to perform specific tasks on a general purpose computer may be executed.” *Id.* For the reasons explained with respect to [1c], a POSITA would

have understood or at least found it obvious that “native applications” would have been configured to run at approximately the same time and certainly no later than when the “user-mode applications” described by Field would have been configured to run, consistent with the following annotation of Field’s FIG. 3. EX1003, ¶121.



Field discloses that “is desirable to start scanning for malware as early as possible in the boot process,” and that this scanning continues throughout the boot process “until traditional antivirus software is available to protect the computer once the boot process is complete.” EX1004, [0022]-[0023]. From this a POSITA would have understood or at least found it obvious that Field’s scan engine 208 and boot detection module 210 would have been configured to **manage pestware-related events after the computer becomes configured to run native applications**. EX1003, ¶122.

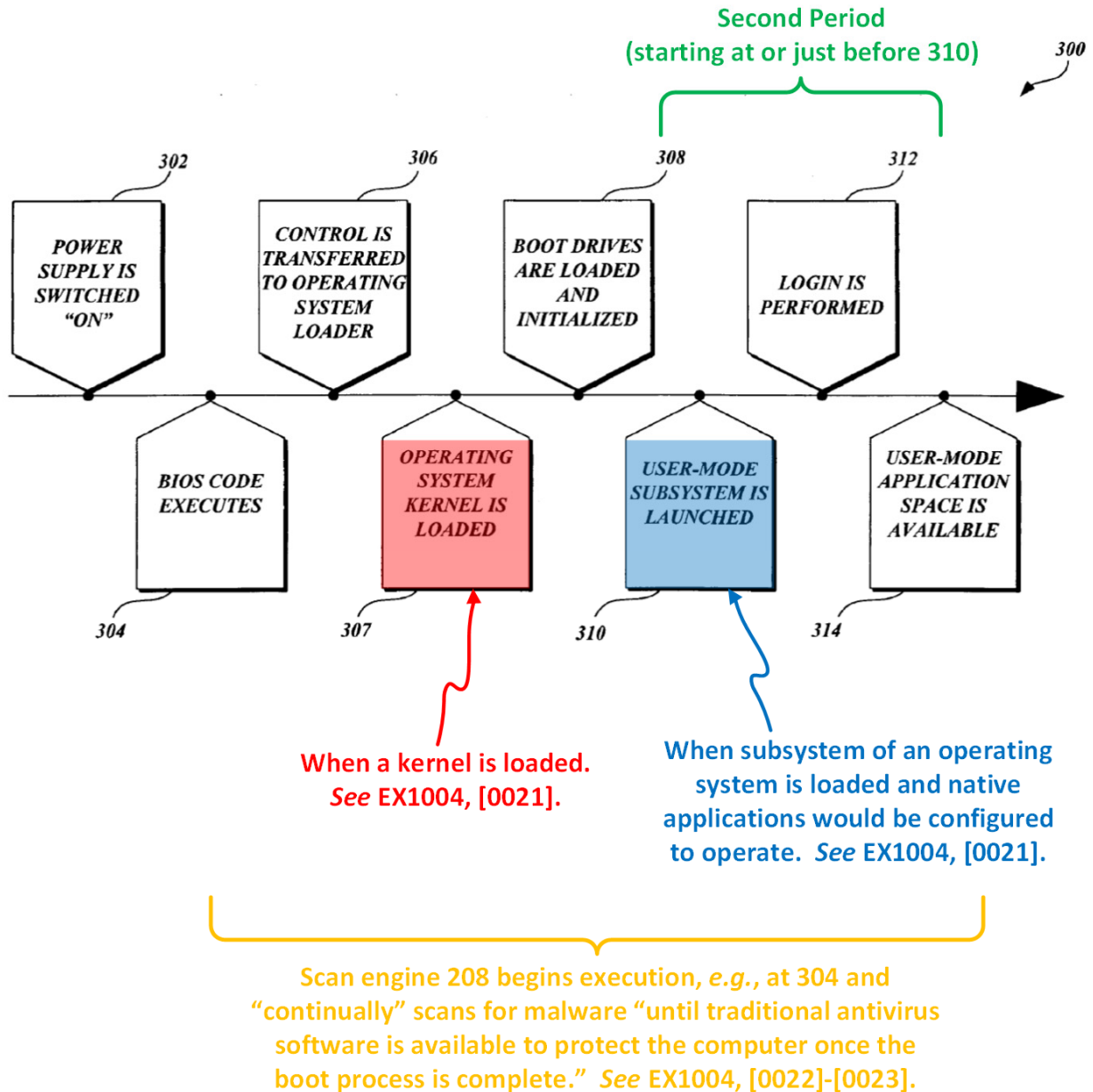
Furthermore, as described with respect to IV.A.4.ii, *supra*, a POSITA would have found it obvious that Field’s scan engine 208 and boot detection module 210 would coordinate with user-mode/application-level components that perform the “traditional antivirus” functionality to which Field refers. *See* EX1004, [0022], [0034]. To this point, both Glick and Shukla describe that the kernel-level components of a malware detection system may continue to run after the boot sequence to coordinate with the application-level components of the system and extend the functionality of the application-level components during normal computer operation. EX1003, ¶123. For example, Glick describes that its kernel mode driver 106B is “loaded prior to all user mode software applications during boot up of host computer system 102” (*see* EX1007, 6:23-32) and that it continues to run after the

boot sequence to assist the user mode application 106A to detect and mitigate malware (*see* EX1007, 12:50-13:44, 18:6-33 (describing various “Request Kernel Mode Assistance” operations)). EX1003, ¶123. From these teachings a POSITA would have found it obvious that Field’s scan engine 208 and boot detection module 210 would have been configured to continue executing and managing pestware-related events after the boot sequence and during the normal operation of the computer. EX1003, ¶123.

(v) Claim 5

[5]

As discussed in 1[c] *supra*, Field teaches operating ***while the boot drivers are initiated***. As discussed *supra*, Field teaches that its scan engine “may be loaded into memory” at “any number of different locations in the timeline” and may “begin executing at events 304, 306, or 310.” EX1004, [0022]-[0023]. This is illustrated in the following annotation of FIG. 3. EX1003, ¶125.



As shown in the above annotation and described in Field, boot drivers are initiated during event 308, which is after event 304 when the scan engine 208 and boot detection module 210 would have begun executing. See EX1004, FIG. 3,

[0021]-[0023]; EX1003, ¶126. Accordingly, in the Field-Glick-Shukla combination **the monitoring includes monitoring the boot sequence while boot drivers are initiated.** EX1003, ¶126.

(vi) **Claim 6**

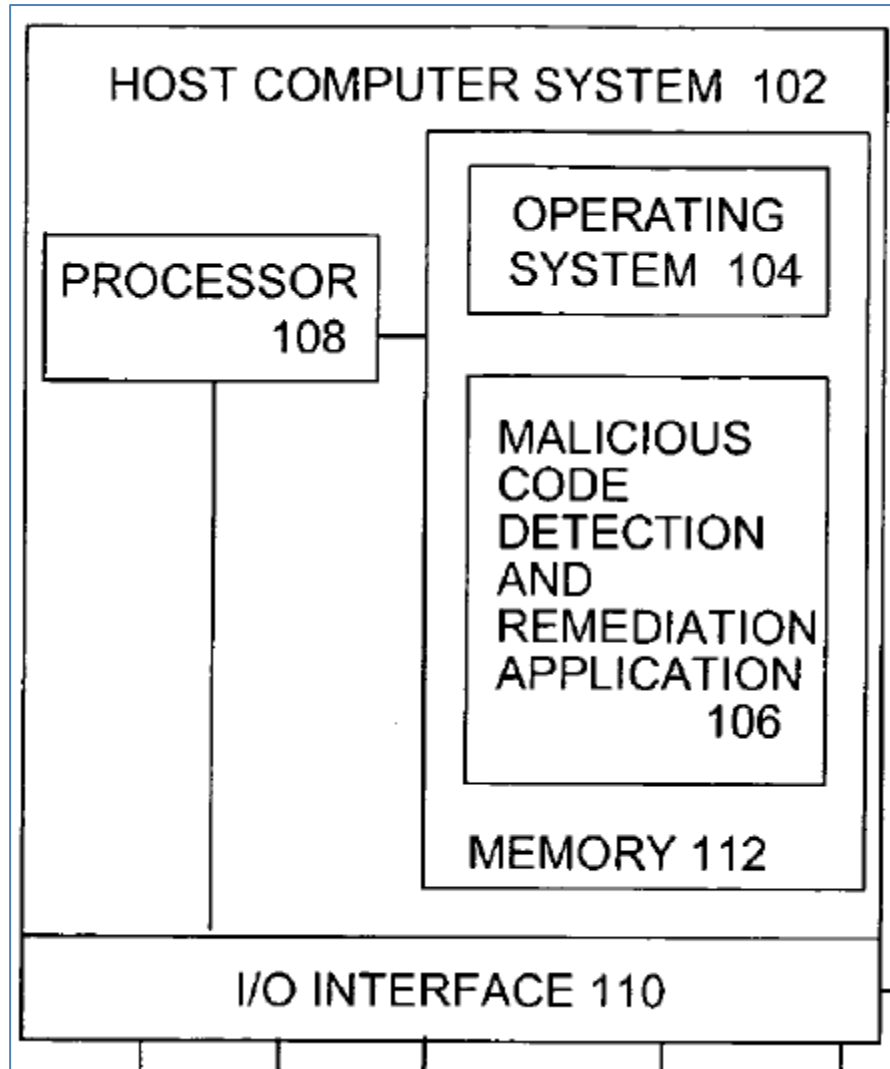
6[pre]

See 1[pre].

6[a]

Field describes that its system operates on a typical computer 200 that “may be any one of a variety of devices including, but not limited to, personal computing devices . . . or other electronic devices having some type of memory.” EX1004, [0014]. Field describes that these types of computers typically included “a keyboard, a mouse, a printer, a display, *CPU*, memory, etc.” *Id.* Thus, Field describes that the scan engine 208 and a boot detection module 210 are configured to run on any typical computer 200, and that computer would include a processor (e.g., a CPU). *Id.*; EX1003, ¶130.

Indeed, Glick’s Figure 1 similarly shows a “malicious code detection and remediation application” on a host computer system with a central processing unit (CPU) 108. EX1007, FIG. 1, 2:3-13.

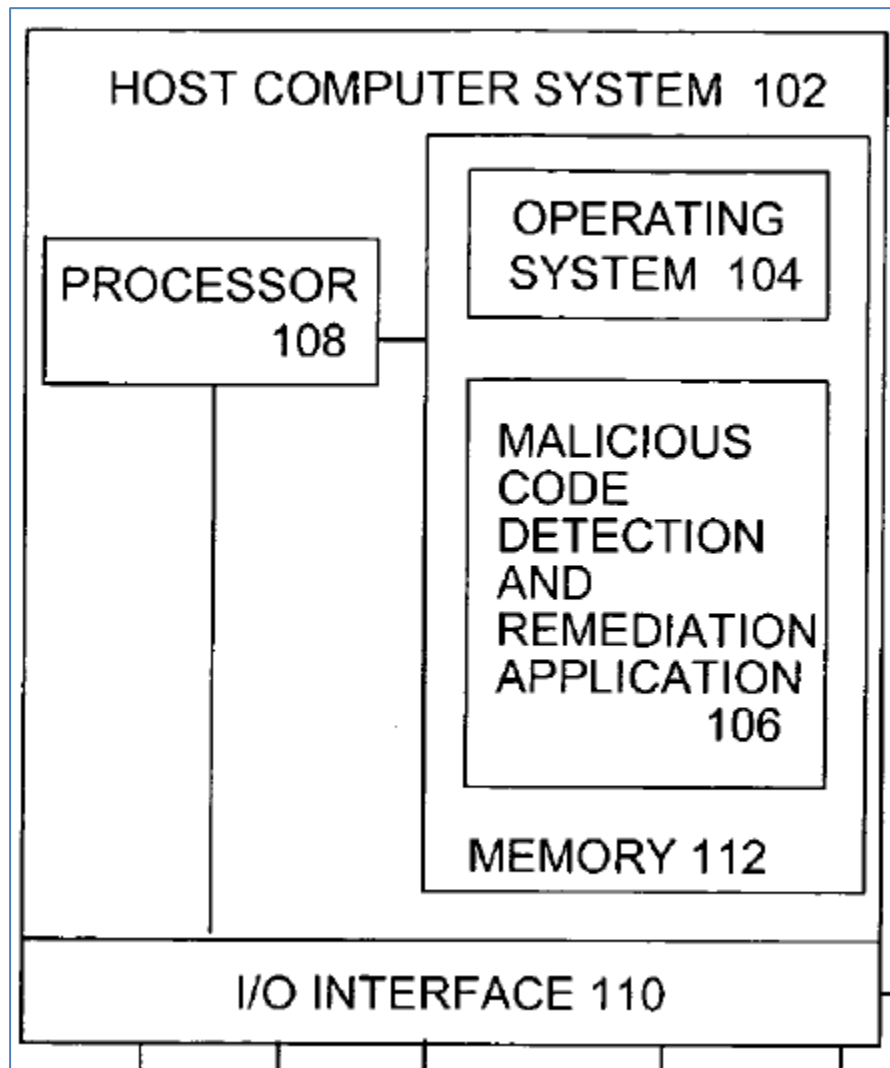


6[b]

As noted above, Field describes that its system operates on a typical computer 200 that “may be any one of a variety of devices including, but not limited to, personal computing devices . . . or other electronic devices having some type of memory.” EX1004, [0014]. Field describes that these types of computers typically included “a keyboard, a mouse, a printer, a display, CPU, *memory*, etc.” *Id.*;

EX1003, ¶132. Indeed, Field teaches that “the boot detection module causes the scan engine 208 to be loaded into *memory*.” EX1004, [0016].

Glick’s Figure 1 similarly shows a “malicious code detection and remediation application” on a host computer system with a memory 112, and that the “malicious code detection and remediation application 106 is typically stored in memory 112 of host computer system 102 and executed on host computer system 102.” EX1007, FIG. 1, 2:3-13, 3:8-11.

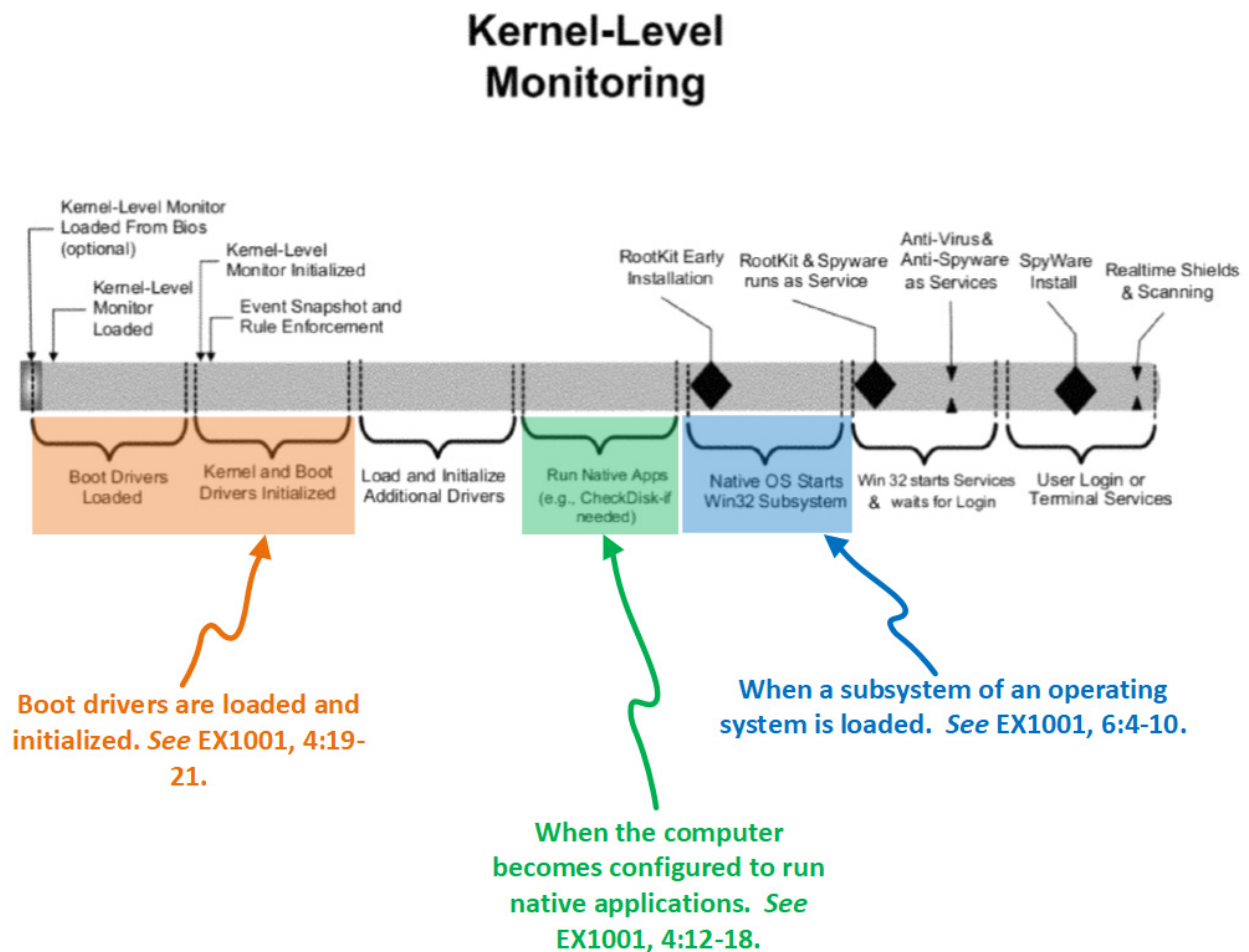


6[b.i]

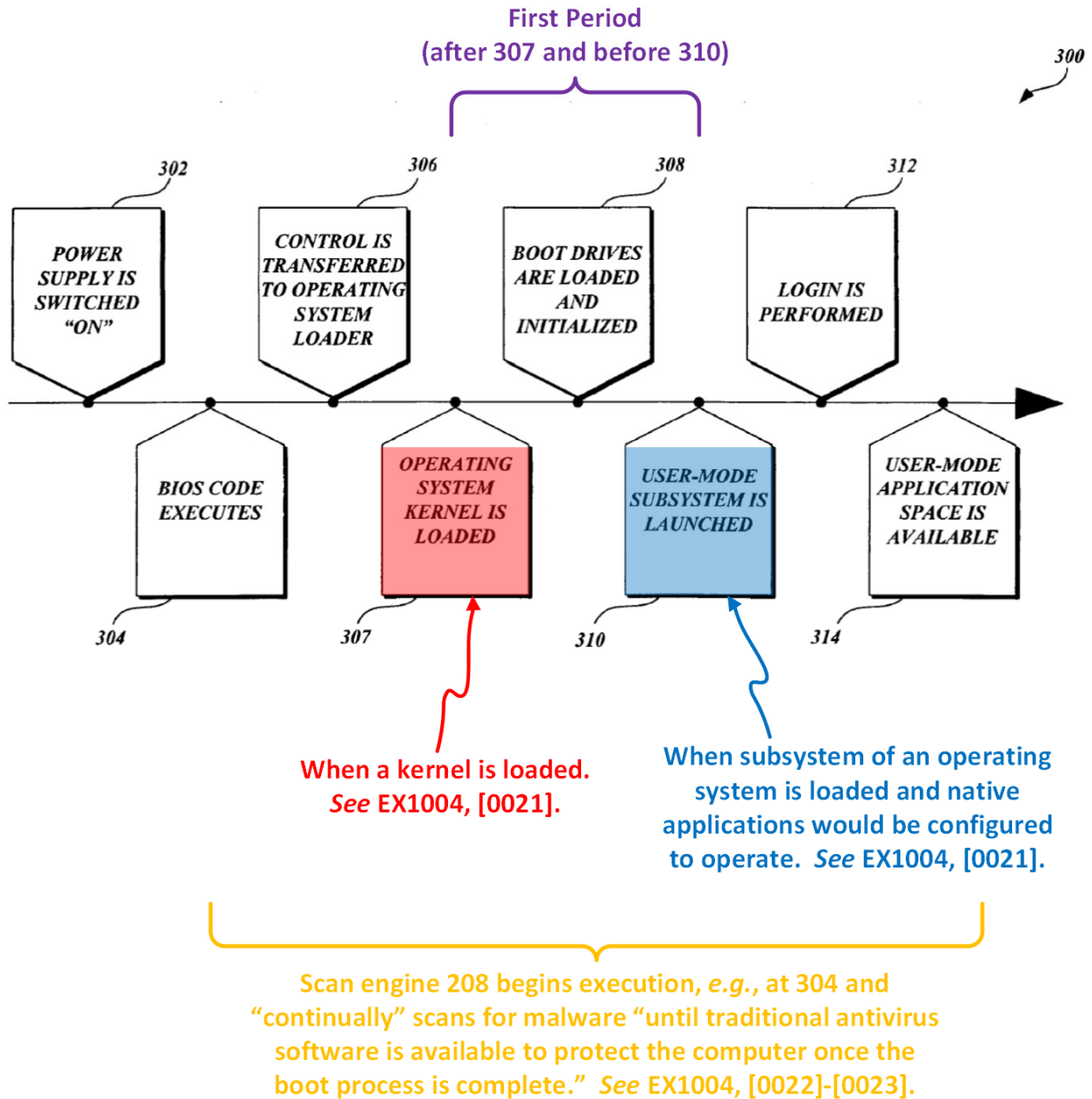
See 1[a]

6[b.ii]

See 1[b]. Notably, this limitation specifies that the first period is “before a WIN32 subsystem is loaded,” where as 1[b] specifies that the first period is “before a sub-system of an operating system is loaded.” As described with respect to 1[b], the sub-system of an operating system is loaded at event 310. *See* EX1004, [0021]; EX1003, ¶135.



Field does not explicitly describe in its timeline when the WIN32 subsystem is loaded, but a POSITA would have understood it to be at or around the same time as event 310. EX1003, ¶136. As described with respect to 1[c], the timeline in FIG. 4 of the '505 Patent describes effectively the same sequence of events in a standard boot process and the timeline in FIG. 3 of Field. EX1003, ¶136. The methods and systems in each of the '505 Patent and Field are described as being operable on general purpose computers that run a variety of operating systems, including Microsoft Windows. EX1001, 3:5-8 (“And if implemented in software, the modules can be designed to operate on any type of computer system including WINDOWS and Linux-based systems”); EX004, [0015] (“The operating system 204 illustrated in FIG.2 may be a general-purpose operating system such as a Microsoft® operating system, UNIX® operating system, or Linux® operating system.”); EX1003, ¶136. Thus, while the exact events indicated on each of the timelines are not identical, a POSITA would nevertheless have understood the two timelines as depicting the same general boot process flow, which would have been specific to a given operating system and computer configuration and independent of the monitoring system/method. EX1003, ¶136. Based on the '505 Patent description of when the WIN32 subsystem is loaded, a POSITA would have understood this to be at or around the same time as event 310 in Field's FIG. 3. *Compare* EX1001, 6:4-10 to EX1004, [0021]; EX1003, ¶136.



Thus, as shown in the above annotation of FIG. 3, where a POSITA configured the scan engine 208 to begin executing at events 304 or 306 per the explicit teachings of Field, the scan engine 208 would be executing (e.g., managing pestware-related events by scanning for malware and removing or quarantining identi-

fied malware) during a **first period** that is **before the computer becomes configured to run native applications, before a WIN32 subsystem is loaded (event 310)**, and **after a kernel is loaded (event 307)**. EX1003, ¶137.

6[b.iii]

See 1[c]

6[b.iv]

See 1[d]

6[b.v]

See 1[e]

6[b.vi]

See 1[f]

6[b.vii]

See 1[g]

(vii) Claim 7

[7]

See [2]

(viii) Claim 8

[8]

See [3]

(ix) Claim 9

[9]

See [4], [1c]

(x) **Claim 10**

[10]

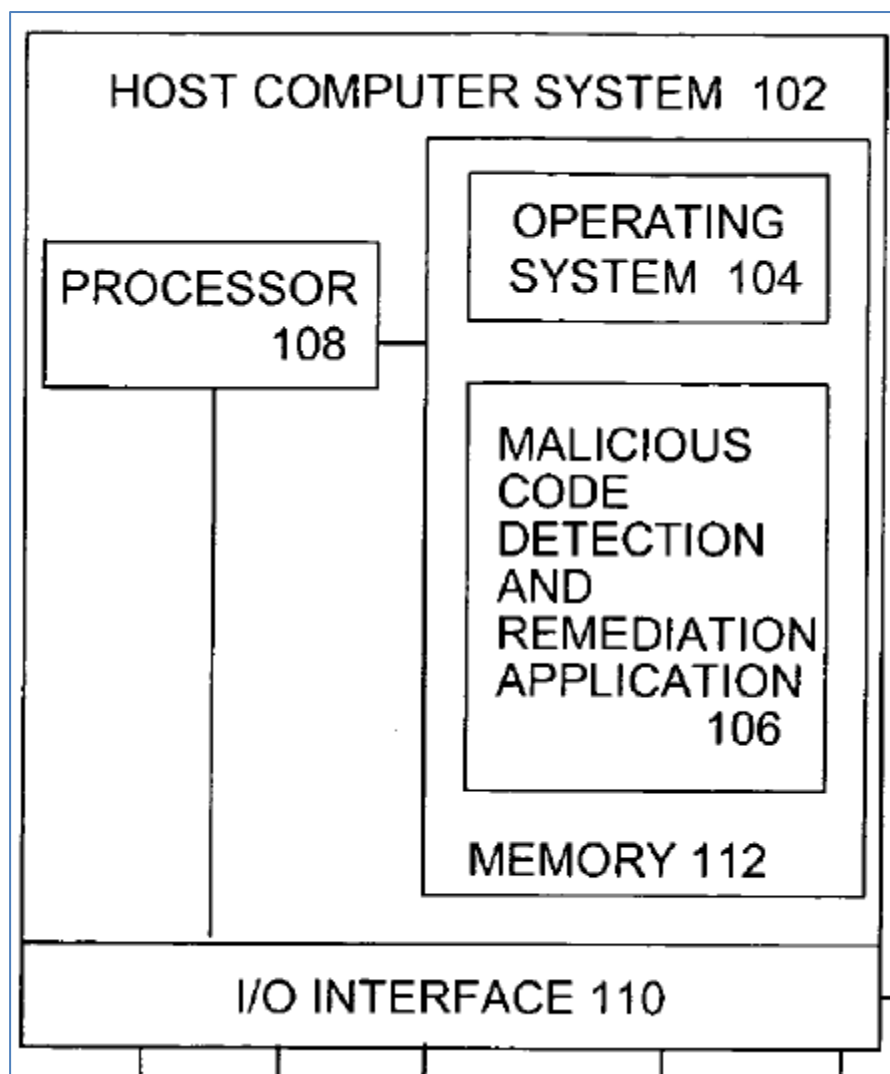
See [5], [1c]

(xi) **Claim 11**

11[pre]

Field describes that its system operates on a typical computer 200 that “may be any one of a variety of devices including, but not limited to, personal computing devices . . . or other electronic devices having some type of memory.” EX1004, [0014]. Field describes that these types of computers typically included “a keyboard, a mouse, a printer, a display, CPU, *memory*, etc.” *Id.* Indeed, Field teaches that “the boot detection module causes the scan engine 208 to be loaded into *memory*.” EX1004, [0016]. A POSITA would have understood or at least found it obvious that Field’s memory was a non-transitory tangible computer-readable storage medium comprising program instructions. EX1003, ¶153.

Glick’s Figure 1 similarly shows a “malicious code detection and remediation application” on a host computer system with a memory 112, and that the “malicious code detection and remediation application 106 is typically stored in memory 112 of host computer system 102 and executed on host computer system 102.” EX1007, FIG. 1, 2:3-13, 3:8-11. A POSITA would have understood or at least found it obvious that Glick’s memory was a non-transitory tangible computer-readable storage medium comprising program instructions. EX1003, ¶154.



11[a]

See 1[a]

11[b]

See 1[b] and 6[b.ii].

11[c]

See 1[c]

11[d]

See 1[d]

11[e]

See 1[e]

11[f]

See 1[f]

11[g]

See 1[g]

(xii) Claim 12

[12]

See [2]

(xiii) Claim 13

[13]

See [3]

(xiv) Claim 14

[14]

See [4]

(xv) Claim 15

[15]

See [5]

(xvi) Claim 16

[16]

As described with respect to [1f], a POSITA would have understood that iteratively observing new behavior and assessing it would have allowed the system

of the Field-Glick-Shukla combination to better adapt to new threats by allowing for the creation of new behavior rules and the modification of old behavior rules. *See* EX1005, [0078]; EX1006, 18; EX1003, ¶171. As Shukla recognized, using rigid signature-based detection techniques was “ineffective” (*see* EX1005, [0078]; EX1006, 18) and that using rigid behavior rules leads to a malware detection system that is either overly burdensome on the user or overly permissive to malware (*see* EX1005, [0119]-[0123]; EX1006, 25-26). EX1003, ¶171. Thus, Shukla teaches the benefits of flexibility in the creation, maintenance, and application of behavior rules in order to adapt to new malware threats. From these teachings, a POSITA would have found it obvious for the Field-Glick-Shukla combination to **modify[] the set of behavior rules so as to prevent the pestware related events.** EX1003, ¶171.

As described with respect to claim 2, given that the application-level components of these types of malware detection systems were known to have broader processing and remediation capabilities (*see, e.g.*, EX1004, [0034]), a POSITA would have understood or at least found it obvious that the user-mode/application-level components of the Field-Glick-Shukla combination would have been configured to compile the behavior rules. EX1003, ¶172. Indeed, this is consistent with Glick’s general descriptions of its user mode application 106A performing the primary analysis and coordination functions, and calling upon the kernel mode driver

106B for specific functional assistance. *See, e.g.*, EX1007, 12:50-13:44, 18:6-33; EX1003, ¶172. Thus, in the Field-Glick-Shukla combination, a POSITA would have understood or at least found it obvious that user-mode/application-level components (i.e., pestware management engine) would have managed the set of behavior rules, including the modification of those rules based on the mathematical/statistical risk analysis described by Shukla. EX1003, ¶172 (citing EX1005, [0110]-[0118]).

As described in Section IV.A.4.iii and with respect to [1b] and [1c], *supra*, a POSITA would have found it obvious that Field’s scan engine 208 and a boot detection module 210—which together operate as a kernel-level monitor (*see* EX1004, [0005], [0017]; EX1007, 6:23-32)—would utilize the above-described behavior rules to **manage[] pestware-related events in the first and second periods in the boot sequence**. EX1003, ¶173. Field describes that “the scan engine may *search for suspicious activities* such as jump instructions in unexpected locations, hidden processes, references to memory addresses that are outside of a range allocated to the operating system, and the like” and that “[a]ntivirus software *may identify activity* that could be characteristic of malware.” EX1004, [0017], [0027]. In one example, Field describes that “[r]emoving an entry from a process table or a similar data structure may be *a strong heuristic indicator* that a RootKit is pre-

sent.” EX1004, [0017]. While Field does not specifically describe this functionality as the use of behavior rules to managing pestware-related events, a POSITA would nonetheless have understood or at least found it obvious that Field’s search for suspicious activities would have included the use of behavior rules based on the teachings of Shukla. EX1003, ¶173.

Shukla describes that its kernel-level component “hooks into kernel-mode API function calls or even unexported API calls and monitors various components of the kernel.” EX1005, [0041]. Shukla teaches that the kernel-level component enforces a set of behavior rules “blocking, permitting, or monitoring the API function calls.” EX1005, [0040], *see also id.*, [0043]-[0057]. Thus, in the Field-Glick-Shukla combination, a POSITA would have found it obvious that Field’s scan engine 208 and a boot detection module 210, modified to utilize behavior rules based on the teachings of Shukla, were a kernel-level monitor that utilize behavior rules to manage pestware-related events in the first and second periods in the boot sequence. EX1003, ¶174.

(xvii) Claim 17

[17]

As described with respect to [1g], *supra*, the Field-Glick-Shukla combination scans data in a registry of the computer for pestware during the second period in the boot sequence. EX1003, ¶176. Shukla describes that, in “Learn mode,” the

analysis performed by the kernel-mode portion of the sandbox may be used to generate and test the behavior rules employed by Shukla's overall malware detection system. *See* EX1005, [0110]-[0118]; EX1006, 24-25; EX1003, ¶176. From these disclosure a POSITA would have understood or at least found it obvious that **modifying the set of behavior rules based on the scanned registry so as to prevent the pestware-related events**, because Shukla teaches that registry scanning is one of the analyses performed by Shukla's kernel module, and a POSITA would have understood or at least found it obvious to use this analysis during "Learn mode" to develop the behavior rules. *Id.*

(xviii) **Claim 18**

[18]

Field teaches monitoring **events associated with... loading pestware, executing pestware, and one or more behaviors of pestware**. For example, Field teaches that a scan engine starts executing early in the boot process so that "subsequently initialized boot drivers are scanned for malware as they are loaded into memory." EX1004, [0030]. A POSITA would have understood or at least found obvious from this teaching that pestware being loaded into memory is a recognized event. EX1003, ¶178. Field also teaches "one or more" other "behaviors of pestware," when it states that "some malware implement self-preservation techniques in which the malware's resources (e.g. files, processes, entries in configuration

files, and the like) are monitored. When an attempt to remove the malware is identified, functionality designed to preserve the malware’s resources and maintain the infection are performed.” EX1004, [0034]; EX1003, ¶178.

Similarly, Glick teaches monitoring **pestware-related events associated with... loading pestware**. EX1003, ¶179. Glick teaches that “In MALICIOUS CODE DETECTED check operation 610, user mode application 106A evaluates the list of running processes and/or dlls loaded into each running process 608 and determines whether malicious code is detected on host computer system 102. EX1007,11:7-31.

(xix) Claim 19

[19]

See [18]

(xx) Claim 20

[20]

See [18].

V. PTAB DISCRETION SHOULD NOT PRECLUDE INSTITUTION

Institution is consistent with the Director’s recent guidance on applying the *Fintiv* Factors. *Apple Inc. v. Fintiv, Inc.*, IPR2020-00019, Paper 11 (PTAB Mar. 20, 2020) (precedential) (“*Fintiv I*”); EX1014 (“*Director’s Memo*”).

A. The *Fintiv* Factors Favor Institution—§314(a)

1. Factor 1

Parallel litigation is ongoing. While no stay has been requested, Petitioner may seek a stay if institution is granted, thus favoring institution. At worst, this factor is neutral because the Board “will not attempt to predict” how the district court will proceed. *Sand Revolution II, LLC v. Continental Intermodal Group-Trucking LLC*, IPR2019-01393, Paper 24 at 7 (PTAB Jun. 16, 2020).

2. Factor 2

“This factor looks at the proximity of the trial date to the date of [FWD] to assess the weight to be accorded a trial date set earlier than the expected [FWD] date.” *Shenzhen Carku Tech. Co., Ltd. v. The Noco Co.*, IPR2020-00944, Paper 20 at 61 (PTAB Nov. 12, 2020). Understanding “scheduled trial dates are unreliable and often change,” the *Director’s Memo* dictates “[a] court’s scheduled trial date...is not by itself a good indicator of whether the district court trial will occur before the statutory deadline for a [FWD].” *Samsung Elec.’s Co., Ltd. et al. v. Staton Techiya, LLC*, IPR2022-00302, Paper 13, at 10 (PTAB July 11, 2022). When analyzing this factor, “the parties [may] rely on time-to-trial statistics” along with “additional supporting factors such as the number of cases before the judge in the parallel litigation and the speed and availability of other case dispositions.” *Id.*, 10-12.

Based on the most recent WDTX statistics, the “actual trial date” is likely to be held mid-January, 2025—28.2 months from September 8, 2022, the date the first Amended Complaint was filed asserting the ’505 Patent against CrowdStrike. EX1015, 37. While the District Court entered a schedule with an August 19, 2024 trial date assigned to each defendant in five separate lawsuits that have been consolidated for pretrial issues, PO has admitted that “median time-to-trial statistics are more appropriate” for consideration under this factor. IPR2023-00289, Paper 6, at 13; EX1012, 8. Either way, because the court’s consolidation is limited to pretrial issues, separate trials will be required, and trial cannot proceed on August 19, 2024 for all defendants.

The Board’s FWD deadline in this case is around December, 2024, a month before a trial would occur based on the time-to-trial statistics this Board should apply to its analysis. However, “additional supporting factors” are at play weighing towards a longer than average filing-to-trial window. First, the District Court litigation involves thirty-nine total patents, including twelve asserted by PO against Petitioner and two counterclaim infringement allegations brought by Petitioner. As PO previously quoted, U.S. Magistrate Judge Gilliland explained in a December 9, 2022 hearing that this is “a phenomenal case that I’m hoping one day maybe people will write songs about because it is about as complicated as you’re going to get with patent cases.” IPR2023-00289, Paper 6, at 2.

To accommodate the significant number of allegations, the Court scheduled three Markman hearings, the first was conducted on March 7, 2023, the second was recently held on May 3, 2023 relating to the Counterclaim patents, and the final Markman is scheduled for July 25, 2023, which will involve the Challenged Patent. EX1013, 3, 5, 6. Accordingly, with multiple Markman hearings and a multitude of claims involved in a phenomenally complex case, the “actual trial date” is much more likely to be delayed until later, commensurate with the median time to trial (or later) and after FWD here.

Moreover, the '505 Patent was added to the litigation in September, 2022, as one of six additional asserted patents, introduced over six months after the original six patents were asserted against Petitioner. The '505 Patent, and other late asserted patents, were shoehorned into an already ongoing litigation, resulting in a shortened window to trial. Petitioner acted diligently in filing its Petition against this newly asserted patent, and any shortened window to trial for these patents is the result of PO's own litigation strategy.

In sum, the Board's FWD deadline in this case falls before a trial would occur based on WDTX time-to-trial statistics. But the additional complicating factors at issue here suggest trial will occur even later in time than the statistics suggest, weighing heavily in favor of institution.

3. Factor 3

This factor favors institution because Petitioner worked diligently to overcome the challenging circumstances of the Litigation and file this Petition three months ahead of the statutory time bar. *Fintiv I*, 11. Beyond claim construction, the parties have invested little in the parallel proceeding. The first and second Markman hearings were just recently conducted on March 7, 2023 and May 3, 2023, respectively. The final Markman is not scheduled until July 25, 2023. Fact discovery just opened on April 18, 2023 and is not set to conclude until January 18, 2024. Expert discovery does not close until March 12, 2024 and dispositive motions are not due until May 7, 2024. This factor weighs in favor of institution.

4. Factor 4

Petitioner asks the Board to consider the unique challenges raised in the Petition. *See Fintiv*, 12-13. Petitioner stipulates that, if the Board institutes *inter partes* review, Petitioner will not seek resolution in the District Court of any ground of invalidity for the '505 Patent that utilizes Field, Shukla, and Glick—the prior art references relied upon in the grounds of the instant petition. Petitioner's proposal is comprehensive and addresses the concern of narrowing the issues, as this stipulation ensures there is no overlap between the '505 Patent invalidity theories before the Board and at issue in the District Court. Accordingly, this factor weighs heavily in favor of institution.

5. Factor 5

Petitioner and Patent Owner are parties to the Litigation. However, members of the Board have noted *Fintiv* addresses only the scenario in which the petitioner is unrelated to a defendant in a parallel proceeding, finding this should weigh against denying institution, but that *Fintiv* “says nothing about situations in which the petitioner is the same as, or is related to, the district court defendant.” *Cisco Sys., Inc. v. Ramot at Tel Aviv Univ. Ltd.*, IPR2020-00122, Paper 15 at 10-11 (PTAB May 15, 2020) (APJ Crumbley, dissenting).

6. Factor 6

“[W]here the PTAB determines that the information presented at the institution stage presents a compelling unpatentability challenge, that determination alone demonstrates that the PTAB should not discretionarily deny institution under *Fintiv*.” *Hewlett Packard Enterprise Co., et al. v. Biljco LLC*, IPR2022-00420, Paper 17, at 6 (PTAB July 12, 2022). The strength of Petitioner’s proposed grounds presented above weighs strongly in favor of institution. *Id.*

B. § 325(d) Favors Institution

Field, Glick, and Shukla were not considered or cited in the prosecution of the ’505 Patent, and consequently, each is being considered for the first time in this IPR proceeding. Thus, none of the Grounds involve the same or substantially the same prior art or arguments previously presented to the Office. 35 U.S.C. § 325(d).

VI. FEES

CrowdStrike, Inc. authorizes the Patent and Trademark Office to charge Deposit Account No. 06-1050 for the fee set in 37 C.F.R. § 42.15(a) and for any additional fees.

VII. CONCLUSION

Petitioner requests cancellation of all Challenged Claims.

VIII. MANDATORY NOTICES UNDER 37 C.F.R § 42.8(a)(1)

A. Real Party-In-Interest Under 37 C.F.R. § 42.8(b)(1)

CrowdStrike, Inc. and CrowdStrike Holdings, Inc. are the real parties-in-interest.

B. Related Matters Under 37 C.F.R. § 42.8(b)(2)

Petitioner is not aware of any, reexamination certificates or petitions for *inter partes* review for the '505 Patent. The '505 Patent is the subject of the following civil actions: *Webroot, Inc. and OpenText Inc. v. CrowdStrike, Inc. and CrowdStrike Holdings, Inc.*, Case 6:22-cv-00241 (WDTX), *Webroot, Inc. and OpenText Inc. v. Sophos Limited*, 6-22-cv-00240 (WDTX), and *Webroot, Inc. and OpenText Inc. v. TrendMicro, Inc.*, Case 6:22-cv-00239, filed on March 4, 2022. The '505 Patent is also subject to a terminal disclaimer, matching its term to U.S. Patent No. 8,190,868 (the patent that resulted from the parent application to the '505 patent).

C. Lead And Back-Up Counsel Under 37 C.F.R. § 42.8(b)(3)

CrowdStrike provides the following designation of counsel.

Lead Counsel	Backup counsel
W. Karl Renner, Reg. No. 41,265 Fish & Richardson P.C. 60 South Sixth Street Suite 3200 Minneapolis, MN 55402 Tel: 202-783-5070 Fax: 877-769-7945 Email: IPR54971-0004IP1@fr.com	David Holt, Reg. No. 65,161 Daniela Glaser, Reg. No. 78,905 Fish & Richardson P.C. 60 South Sixth Street Suite 3200 Minneapolis, MN 55402 Tel: 202-783-5070 Fax: 877-769-7945 IPR54971-0004IP1@fr.com

D. Service Information

Please address all correspondence and service to the address listed above.

Petitioner consents to electronic service by email at IPR54971-0004IP1@fr.com

(referencing No. 54971-0004IP1 and cc'ing PTABInbound@fr.com).

Respectfully submitted,

Dated June 30, 2023

/W. Karl Renner/

W. Karl Renner, Reg. No. 41,265
David Holt, Reg. No. 65,161
Daniela Glaser, Reg. No. 78,905
Fish & Richardson P.C.
60 South Sixth Street
Suite 3200
Minneapolis, MN 55402
T: 202-783-5070
F: 877-769-7945

(Control No. IPR2023-01159)

Counsel for Petitioner

CERTIFICATION UNDER 37 CFR § 42.24

Under the provisions of 37 CFR § 42.24(d), the undersigned hereby certifies that the word count for the foregoing Petition for *Inter Partes* Review totals 13,965 words, which is less than the 14,000 allowed under 37 CFR § 42.24.

Dated June 30, 2023

/W. Karl Renner/
W. Karl Renner, Reg. No. 41,265
David Holt, Reg. No. 65,161
Daniela Glaser, Reg. No. 78,905
Fish & Richardson P.C.
60 South Sixth Street
Suite 3200
Minneapolis, MN 55402
T: 202-783-5070
F: 877-769-7945

Counsel for Petitioner

CERTIFICATE OF SERVICE

Pursuant to 37 CFR §§ 42.6(e)(4)(i) *et seq.* and 42.105(b), the undersigned certifies that on June 30, 2023, a complete and entire copy of this Petition for *Inter Partes* Review, Power of Attorney, and all supporting exhibits were provided, via Federal Express, to the Patent Owner by serving the correspondence address of record as follows:

Sprinkle IP Law Group/OPEN
1301 W. 25th Street, Suite 400
Austin, TX 78705
UNITED STATES

/Michael Stanwyck/
Michael Stanwyck
Fish & Richardson P.C.
60 South Sixth Street, Suite 3200
Minneapolis, MN 55402
(202) 783-5070