

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CROWDSTRIKE, INC,
Petitioner,

v.

OPEN TEXT INC.,
Patent Owner.

IPR2022-01522
Patent 10,284,591 B2

Before PATRICK M. BOUCHER, MICHELLE N. WORMMEESTER, and
AARON W. MOORE, *Administrative Patent Judges*.

BOUCHER, *Administrative Patent Judge*.

DECISION
Granting Institution of *Inter Partes* Review
35 U.S.C. § 314(a)

CrowdStrike, Inc. (“Petitioner”) filed a Petition pursuant to 35 U.S.C. §§ 311–319 to institute an *inter partes* review of claims 1, 3, 4, 9, 11, 12, and 17 of U.S. Patent No. 10,284,591 B2 (Ex. 1001, “the ’591 patent”).

Paper 1 (“Pet.”). Open Text Inc. (“Patent Owner”) filed a Preliminary Response. Paper 8 (“Prelim. Resp.”).

We have authority under 35 U.S.C. § 314 and 37 C.F.R. § 42.4. Applying the standard set forth in 35 U.S.C. § 314(a), which requires demonstration of a reasonable likelihood that Petitioner would prevail with respect to at least one challenged claim, we grant the Petition and institute an *inter partes* review. The Board has not made a final determination regarding the patentability of any claim.

I. BACKGROUND

A. The ’591 Patent

1. Overview

The ’591 patent describes “anti-exploit systems and methods” in the context where “[a]n exploit is a piece of code, software, data, or a sequence of commands that takes advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or any other electronic component.” Ex. 1001, 1:12–16, 1:33–35. Figure 3 of the ’591 patent is reproduced below.

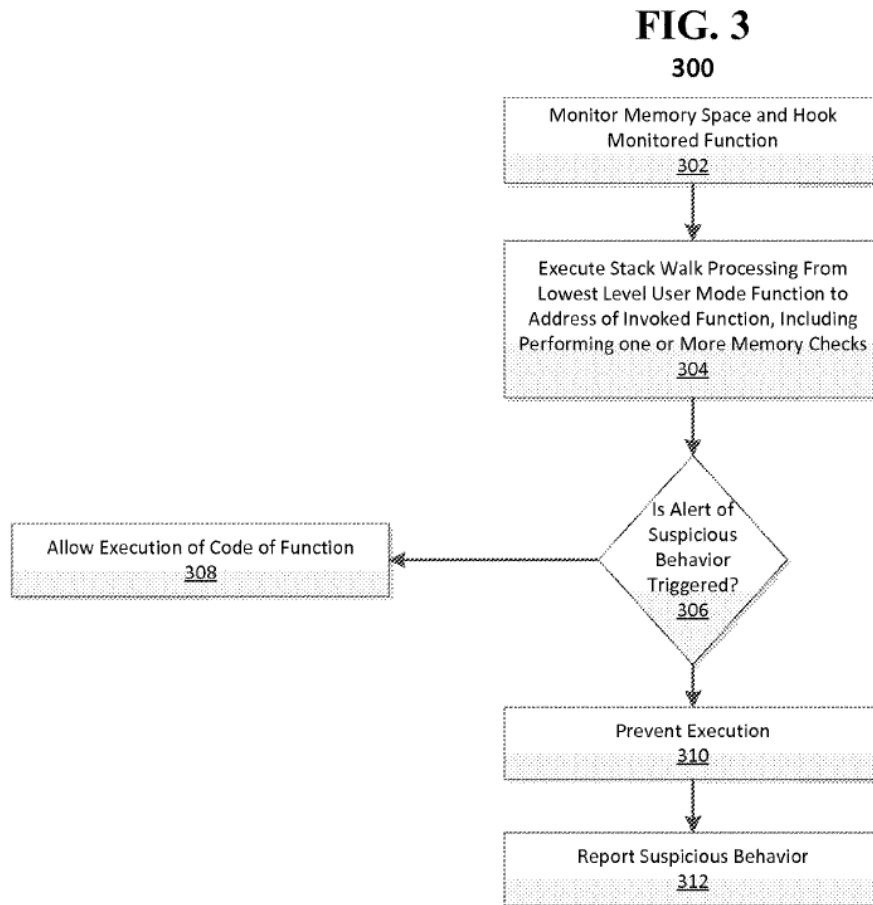


Figure 3 illustrates a method “for implementation of anti-exploit systems and methods.” *Id.* at 10:5–7.

Method 300 begins at operation 302, “where a memory space of a process is monitored for execution of at least one of a plurality of functions,” which may include “any functions that attempt execution of another process or loading of a module into a process memory, implementable by/in other operation systems.” *Id.* at 5:10–15, 10:15–17. “The list of monitored functions may change perpetually and can be updated (e.g., by an administrator or software update) at any time to improve the robustness of the functions monitored and improve security against software exploits.” *Id.* at 5:15–19.

The '591 patent specifically notes an example in which operation 302 “monitors functions associated with memory management in the memory space by a user mode function (e.g., a call initiated from a user mode).” *Id.* at 10:18–21. And the patent additionally notes an example of “applying, upon detecting a function call of one of the plurality of functions in the monitored memory space, a function hooking technique to hook a lowest level user mode function of a called process (e.g., API).” *Id.* at 10:21–26. Monitoring operation 302 “may further comprise detecting an attempt by unidentifiable code to allocate a new executable page of memory and monitoring the allocated new executable page of memory.” *Id.* at 10:26–29.

When one of the monitored functions is invoked, stack walk processing is performed at operation 304 for an associated stack frame. *Id.* at 10:30–35. This is “a process that walks a stack frame for a hooked function[] from a lowest level user mode function to an address of the invoked function.” *Id.* at 10:35–39. Operation 304 also includes performing memory checks during the stack walk processing, such as identifying an originating address associated with the invoked function. *Id.* at 10:39–44.

Decision operation 306 determines whether a suspicious-behavior alert is triggered based on execution of the memory checks. *Id.* at 10:45–48. This includes the following:

- Is code execution attempted from non-executable memory;
- Is a base pointer of a stack frame identified as being invalid;
- Is invalid stack return address identified for a stack frame;
- Is attempted execution of a return-oriented programming technique detected;
- Is the base pointer of a stack frame detected as being outside a current thread stack; and
- Is a return address detected as being inside a virtual memory area.

Id. at 8:18–33. If no suspicious-behavior alert is triggered, code executed associated with the invoked function is allowed at operation 308. *Id.* at 10:54–58. Otherwise, if an alert is triggered, code execution associated with the invoked function is prevented at operation 310, and suspicious behavior may be reported at operation 312. *Id.* at 10:58–65.

2. Illustrative Claim

Challenged independent claim 1 is representative of the challenged claims, and is reproduced below.

1. A computer-implemented method comprising:
 - monitoring a memory space of a process for execution of at least one monitored function of a plurality of functions, wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space;
 - invoking one of the plurality of functions as a result of receiving a call from an application programming instance;
 - executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and
 - performing, during the executing of the stack walk processing before an address of an originating caller function is reached, a memory check for a plurality of stack entries identified during the stack walk processing to detect suspicious behavior, wherein an alert of suspicious behavior is triggered when the performing of the memory check detects at least one of the following:
 - code execution is attempted from non-executable memory,
 - a base pointer is identified as being invalid,
 - an invalid stack return address is identified,
 - attempted execution of a return-oriented programming technique is detected,

the base pointer is detected as being outside a
current thread stack, and
a return address is detected as being inside a
virtual memory area,
wherein when an alert of suspicious behavior is triggered,
preventing execution of a payload for the invoked function
from operating.

Ex. 1001, 13:29–61.

B. Evidence

Petitioner relies on the following references:

Shukla	US 2008/0016339 A1	Jan. 17, 2008	Ex. 1004
Dalcher	US 2013/0047255 A1	Feb. 21, 2013	Ex. 1005
Tosa	US 9,465,936 B2	Oct. 11, 2016	Ex. 1006
Kc	US 7,971,255 B1	June 28, 2011	Ex. 1007

In addition, Petitioner relies on a Declaration by Wenke Lee, Ph.D.
Ex. 1003. Patent Owner relies on a Declaration by Nenad Medvidovic,
Ph.D. Ex. 2001.

C. Asserted Grounds of Unpatentability

Petitioner challenges claims 1, 3, 4, 9, 11, 12, and 17 on the following
grounds. Pet. 13–14.

Claim(s) Challenged	35 U.S.C. § ¹	References
1, 3, 9, 11, 17	103	Shukla, Dalcher
4, 12	103	Shukla, Dalcher, Tosa
1, 9, 17	102(a)(1)	Kc
1, 9, 17	103	Kc

D. Real Parties in Interest

Petitioner identifies CrowdStrike, Inc. and CrowdStrike Holdings, Inc. as real parties in interest. Pet. 70. Patent Owner identifies Open Text Inc. and Webroot LLC as real parties in interest. Paper 7, 1. “Out of an abundance of caution, Patent Owner further identifies the following affiliates: Carbonite, LLC; Open Text Corporation; and Open Text Holdings, Inc.” *Id.*

E. Related Matters

Both parties identify *Webroot, Inc. v. CrowdStrike, Inc.*, No. 6:22-cv-00241 (W.D. Tex.) (“the related litigation”) as a related matter in which the ’591 patent is the subject of a patent-infringement lawsuit filed against Petitioner. Pet. 70; Paper 7, 1. Both parties also identify the following related matters as involving the ’591 patent with different defendants: (1) *Webroot, Inc. v. Trend Micro Inc.*, No. 6:22-cv-00239

¹ The Leahy-Smith America Invents Act (“AIA”), Pub. L. No. 112-29, 125 Stat. 284, 287–88 (2011), amended 35 U.S.C §§ 102 and 103. The ’591 patent claims the benefit of the January 27, 2014, filing date of U.S. Prov. Patent Appl. No. 61/931,722, which Petitioner applies as the effective filing date for the challenged claims. Ex. 1001 at code (60); Pet. 7–8. Patent Owner does not dispute this effective filing date. We accordingly apply the AIA versions of §§ 102 and 103 herein.

(W.D. Tex.); (2) *Webroot, Inc. v. Sophos Ltd.*, No. 6:22-cv-00240 (W.D. Tex.); and (3) *Webroot, Inc. v. AO Kaspersky Lab*, No. 6:22-cv-00243 (W.D. Tex.). Pet. 70; Paper 7, 1. Patent Owner additionally identifies *Webroot, Inc. v. Forcepoint LLC*, No. 6:22-cv-00342 (W.D. Tex.) as another related matter involving the '591 patent. Paper 7, 1.

Patent Owner further identifies certain *inter partes* review proceedings involving patents that it characterizes as “not related to the '591 Patent,” but which are also asserted in the district court litigations identified above, namely IPR2023-00124, IPR2023-00126, IPR2023-00289, and IPR2023-00491. *Id.* at 2.

II. DISCRETION UNDER 35 U.S.C. § 314(a)

As noted above, the '591 patent is the subject of parallel proceedings in the form of the related litigation. *Supra*, § I.E. Patent Owner asks that we exercise our discretion to deny the Petition based on the state of the related litigation. Prelim. Resp. 4–14.

Institution of an *inter partes* review is discretionary. *See* 35 U.S.C. § 314(a) (2018) (stating “[t]he Director *may not* authorize an *inter partes* review to be instituted unless the Director determines that the information presented in the petition . . . shows that there is a reasonable likelihood that the petitioner would prevail with respect to at least 1 of the claims challenged in the petition”) (emphasis added); *Harmonic Inc. v. Avid Tech., Inc.*, 815 F.3d 1356, 1367 (Fed. Cir. 2016) (“[T]he PTO is permitted, but never compelled, to institute an IPR proceeding.”). The advanced state of a parallel district court action may warrant exercising discretion on behalf of the Director to deny a petition for *inter partes* review. *See NHK Spring Co.*

v. Intri-Plex Techs., Inc., IPR2018-00752, Paper 8 at 20 (PTAB Sept. 12, 2018) (precedential) (“*NHK*”); *Apple Inc. v. Fintiv Inc.*, IPR2020-00019, Paper 11 at 5–6, 8 (PTAB March 20, 2020) (precedential) (“*Fintiv*”); Patent Trial and Appeal Board Consolidated Trial Practice Guide (Nov. 2019), 58 & n.2, *available at* <https://www.uspto.gov/TrialPracticeGuideConsolidated> (“Trial Practice Guide”). Whether to exercise such discretion is informed by the Director’s Interim Procedure for Discretionary Denials in AIA Post-Grant Proceedings with Parallel District Court Litigation (“Interim Procedure”).²

We consider the following factors in assessing “whether efficiency, fairness, and the merits support the exercise of authority to deny institution in view of an earlier trial date in the parallel proceeding”:

1. whether the court granted a stay or evidence exists that one may be granted if a proceeding is instituted;
2. proximity of the court’s trial date to the Board’s projected statutory deadline for a final written decision;
3. investment in the parallel proceeding by the court and the parties;
4. overlap between issues raised in the petition and in the parallel proceeding;
5. whether the petitioner and the defendant in the parallel proceeding are the same party; and
6. other circumstances that impact the Board’s exercise of discretion, including the merits.

² Available at https://www.uspto.gov/sites/default/files/documents/interim_proc_discretionary_denials_aia_parallel_district_court_litigation_memo_20220621_.pdf.

Fintiv at 5–6. In evaluating these factors, we “take[] a holistic view of whether efficiency and integrity of the system are best served by denying or instituting review.” *Id.* at 6.

1. Possibility of Stay

A stay of a related proceeding pending resolution of the PTAB trial “allays concerns about inefficiency and duplication of efforts.” *Fintiv* at 6. At this time, no stay has been requested or ordered in the related litigation, although Petitioner states that it “will seek a stay if institution is granted.” Pet. 64; *see also* Prelim. Resp. 5. According to Patent Owner, even if Petitioner does seek a stay, it is unlikely the court will grant one because of the complexity of the related litigation, which includes multiple patents, additional defendants, and various counterclaim allegations. Prelim. Resp. 5–6. Petitioner acknowledges this complexity. Pet. 66.

Although we agree with Patent Owner that a stay appears unlikely even if we institute trial, there has been no actual denial of a stay as contemplated by *Fintiv* to weigh this factor against exercising discretion to deny institution. *See Fintiv* at 6–7. We accordingly treat this factor as neutral.

2. Schedules

According to *Fintiv*, “[i]f the court’s trial date is earlier than the projected statutory deadline, the Board generally has weighed this fact in favor of exercising authority to deny institution.” *Fintiv* at 9. The parties agree that, according to median time-to-trial statistics in the Western District of Texas, where the related litigation is pending, the expected trial date for

the related litigation is mid-July 2024. Pet. 65; Prelim. Resp. 7. This is generally consistent with the currently scheduled trial date of August 19, 2024. See Ex. 2007, 18; Prelim. Resp. 7. Although Petitioner contends that the complexity of the related litigation is likely to delay the trial further, the currently scheduled date is already about four months later than the deadline for the Board to issue a final written decision. See Pet. 66. This time difference is already sufficient for this factor to weigh against exercising discretion to deny institution, and we need not speculate whether a further trial delay in the related litigation is likely.

We accordingly treat this factor as weighing against exercising our discretion to deny institution.

3. Investment in Parallel Proceeding

“[I]f, at the time of the institution decision, the district court has issued substantive orders related to the patent at issue in the petition, this fact favors denial” of the Petition. *Fintiv* at 9–10. According to the current scheduling order in the related litigation, a *Markman* hearing was scheduled for April 17, 2023, and fact discovery was scheduled to have opened on April 18, 2023. Ex. 2007, 6. The parties have otherwise not invested a great deal in the related litigation, with fact and expert discovery scheduled for closure only on January 18, 2024 and March 12, 2024, respectively. *Id.* at 7. Although Patent Owner argues that Petitioner “delayed nearly seven months after receiving Patent Owner’s March 4, 2022 complaint, two and a half months after receiving Patent Owner’s infringement contentions, and shortly after filing its own invalidity contentions before filing its petition,” such

delays are neither particularly short nor particularly long. *See* Prelim. Resp. 10.

We treat this factor as weighing against exercising our discretion to deny institution.

4. *Overlap of Issues*

“[I]f the petition includes the same or substantially the same claims, grounds, arguments, and evidence as presented in the parallel proceeding, this fact has favored denial.” *Fintiv* at 12. “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 ’591 Patent that utilizes *any* prior art reference relied upon in the grounds of the instant petition.”

Pet. 67. Such a stipulation is more restrictive than a *Sand Revolution* stipulation but less restrictive than a *Sotera* stipulation. *See Sand Revolution II, LLC v. Continental Intermodal Group-Trucking LLC*, IPR2019-01393, Paper 24, 11–12 (PTAB June 16, 2020) (informative); *Sotera Wireless, Inc. v. Masimo Corp.*, IPR2020-01019, Paper 12, 13–14 (Dec. 1, 2020) (precedential). Thus, while not rising to the level of a stipulation that the Director’s Interim Procedure indicates would preclude discretionary denial, Petitioner’s stipulation does mitigate concern about overlapping issues with the related litigation. *See* Interim Procedure, 7–8.

We treat this factor as weighing against exercising discretion to deny institution.

5. *Overlap of Parties*

If Petitioner was unrelated to a defendant in the related litigation, that might weigh against exercising discretion to deny institution. *Fintiv* at 13. But here, the parties are the same as in the related litigation. Pet. 67; Prelim. Resp. 11–12. Because trial in the related litigation is scheduled to occur well after the deadline for a Final Written Decision in this proceeding, this factor weighs against exercising discretion to deny institution. See *Huawei Tech. Co. v. WSOU Inv., LLC*, IPR2021-00225, Paper 11 at 14 (PTAB June 14, 2021) (“this factor favors denial if trial precedes the Board’s Final Written Decision and favors institution if the opposite is true”) (internal quotation marks omitted); *Google LLC v. Parus Holdings, Inc.*, IPR2020-00846, Paper 9 at 21 (PTAB Oct. 21, 2020) (“Here, . . . Petitioner is the defendant in the parallel proceeding. This fact could weigh either in favor of, or against, exercising discretion to deny institution, depending on which tribunal was likely to address the challenged patent first.”).

6. *Other Circumstances*

The final factor takes into account any other relevant circumstances, including the merits. “For example, if the merits of a ground raised in the petition seem particularly strong on the preliminary record, this fact has favored institution.” *Fintiv* at 14–15. “[C]ompelling, meritorious challenges will be allowed to proceed at the PTAB even where district court litigation is proceeding in parallel.” Interim Procedure, 3–5. We address the merits of Petitioner’s challenges below. For the reasons we explain, at least some of Petitioner’s challenges meet the “reasonable likelihood” standard that we apply in determining whether to institute *inter partes* review, but do not

meet the “compelling” standard. *See* Pet. 68 (Petitioner asserting that “[t]he strength of Petitioner’s proposed grounds . . . weighs strongly in favor of institution”); Prelim. Resp. 13–14 (Patent Owner asserting that “Petitioner’s prior art grounds contain critical deficiencies and do not demonstrate a reasonable likelihood of—let alone plainly lead to—a conclusion that at least one claim is unpatentable”).

We accordingly treat this factor as neutral.

7. *Assessment*

As discussed above, the second, third, fourth, and fifth *Fintiv* factors weigh against exercising our discretion to deny institution; and the first and sixth factors are neutral. In light of the parties’ relatively modest investment thus far in the related litigation and the likelihood that trial will occur in the related litigation only several months after we reach a Final Written Decision in this proceeding, we decline to exercise our discretion under 35 U.S.C. § 314(a) to deny the Petition.

III. ANALYSIS

A. *Legal Principles*

Petitioner makes both anticipation and obviousness challenges. To establish anticipation under 35 U.S.C. § 102, each and every element in a claim, arranged as recited in the claim, must be found in a single prior art reference. *Net MoneyIN, Inc. v. VeriSign, Inc.*, 545 F.3d 1359, 1369 (Fed. Cir. 2008); *Karsten Mfg. Corp. v. Cleveland Golf Co.*, 242 F.3d 1376, 1383 (Fed. Cir. 2001). While the elements must be arranged in the same way as is recited in the claim, “the reference need not satisfy an *ipsissimis verbis* test.”

In re Gleave, 560 F.3d 1331, 1334 (Fed. Cir. 2009); *In re Bond*, 910 F.2d 831, 832–33 (Fed. Cir. 1990). Identity of terminology between the anticipatory prior art reference and the claim is not required. Prior art references must be “considered together with the knowledge of one of ordinary skill in the pertinent art.” *In re Paulsen*, 30 F.3d 1475, 1480 (Fed. Cir. 1994).

A claim is unpatentable for obviousness under 35 U.S.C. § 103 if the differences between the claimed subject matter and the prior art are “such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains.” *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 406 (2007). The question of obviousness is resolved on the basis of underlying factual determinations, including: (1) the scope and content of the prior art; (2) any differences between the claimed subject matter and the prior art; (3) the level of skill in the art; and (4) when in evidence, objective indicia of nonobviousness, i.e., secondary considerations. *Graham v. John Deere Co.*, 383 U.S. 1, 17–18 (1966).

Additionally, the obviousness inquiry typically requires an analysis of “whether there was an apparent reason to combine the known elements in the fashion claimed by the patent at issue.” *KSR*, 550 U.S. at 418 (citing *In re Kahn*, 441 F.3d 977, 988 (Fed. Cir. 2006) (requiring “articulated reasoning with some rational underpinning to support the legal conclusion of obviousness”)); see *In re Warsaw Orthopedic, Inc.*, 832 F.3d 1327, 1333 (Fed. Cir. 2016) (citing *DyStar Textilfarben GmbH & Co. Deutschland KG v. C.H. Patrick Co.*, 464 F.3d 1356, 1360 (Fed. Cir. 2006)).

B. Level of Ordinary Skill in the Art

In determining whether an invention would have been obvious at the time it was made, we consider the level of ordinary skill in the pertinent art at the time of the invention. *Graham*, 383 U.S. at 17. “The importance of resolving the level of ordinary skill in the art lies in the necessity of maintaining objectivity in the obviousness inquiry.” *Ryko Mfg. Co. v. Nu-Star, Inc.*, 950 F.2d 714, 718 (Fed. Cir. 1991). The “person of ordinary skill in the art” is a hypothetical construct, from whose vantage point obviousness is assessed. *In re Rouffet*, 149 F.3d 1350, 1357 (Fed. Cir. 1998). “This legal construct is akin to the ‘reasonable person’ used as a reference in negligence determinations” and “also presumes that all prior art references in the field of the invention are available to this hypothetical skilled artisan.” *Id.* (citing *In re Carlson*, 983 F.2d 1032, 1038 (Fed. Cir. 1993)).

Petitioner proposes that a person of ordinary skill in the art “would have been a person having a bachelor’s degree in computer science, computer engineering, or an equivalent, as well as four years of industry experience and would have had a working knowledge of host monitoring systems, software security analysis, and dynamic malware analysis.” Pet. 13. Petitioner adds that “[a]dditional graduate education could substitute for professional experience, or significant experience in the field could substitute for formal education.” *Id.* Petitioner supports its articulation of the level of ordinary skill in the art with testimony by Dr. Lee. Ex. 1003 ¶ 30.

Patent Owner’s statement of the level of ordinary skill is generally similar, and supported with testimony by Dr. Medvidovic, who characterizes

the parties’ respective positions as having “minor variations.” Prelim. Resp. 26–27; Ex. 2001 ¶¶ 59–60. According to Patent Owner, a person of ordinary skill in the art “would have had a bachelor’s degree in an accredited program of electrical engineering, computer engineering, computer science, or a similar discipline, and have 2-3 years of practical work or research experience in the general fields of electrical engineering, computer science, networking, communications, and/or device and network security.” Prelim. Resp. 26–27. Patent Owner adds that “[m]ore advanced degrees and/or training in a related discipline can compensate for shorter work experience.” *Id.*

For purposes of this Decision, we adopt Petitioner’s proposal, which we find consistent with the level of skill reflected by the prior art. *See Okajima v. Bourdeau*, 261 F.3d 1350, 1355 (Fed. Cir. 2001) (the prior art may reflect an appropriate level of skill in the art). But we note that we would reach the same ultimate conclusion to institute *inter partes* review if we were to adopt Patent Owner’s proposal. *See* Ex. 2001 ¶ 60 (Dr. Medvidovic testifying that the “minor variations” in the parties’ proposals “would not alter [his] opinions.”).

C. Claim Construction

The Board uses “the same claim construction standard that would be used to construe the claim in a civil action under 35 U.S.C. 282(b), including construing the claim in accordance with the ordinary and customary meaning of such claim as understood by one of ordinary skill in the art and the prosecution history pertaining to the patent.” 37 C.F.R. § 42.100(b)

(2019); *see Phillips v. AWH Corp.*, 415 F.3d 1303, 1312–13 (Fed. Cir. 2005) (en banc).

1. “application programming instance”

Independent claims 1, 9, and 17 recite an “application programming instance.” Ex. 1001, 13:36–37, 14:39–40, 15:43–44. As Petitioner observes, the specification of the ’591 patent does not use this term outside of the claims, but does refer to an “application programming interface” and defines that term with the acronym “API.” Pet. 11–12; *see* Ex. 1001, 3:43–44. The parties agree that the “application programming instance” recited in the independent claims “include[s]” an application programming interface. Pet. 12 (Petitioner asserting that it “is interpreting ‘application programming instance’ to at least include an application programming interface (API)”); Prelim. Resp. 23 (Patent Owner asserting that it “agrees that, while ‘application programming instance’ is not limited to an API, this term would include an API”).)

At this stage, we discern no reason to depart from the parties’ shared understanding, and accordingly construe “application programming instance” as including an application programming interface for purposes of this Decision.

2. “at least one of the following”

Each of independent claims 1, 9, and 17 introduces a list of alternatives by reciting that “an alert of suspicious behavior is triggered when the performing of the memory check detects at least one of the following.” Ex. 1001, 13:45–48, 14:48–51, 16:8–11. Each of the lists

includes entries that correspond to the six “memory check” items of column 204 of Figure 2, with claim 17 additionally reciting “a low level user mode function is attempted to be executed in a cache, wherein the cache is evaluated for an address of the invoked function.” *Id.* at Figure 2, 8:18–33, 13:49–58, 14:52–61, 16:12–24. Petitioner contends that each of these limitations “is a Markush group,” and Patent Owner agrees that “the memory check detects at least one of a list of alternatives.” Pet. 12; Prelim. Resp. 24.

We agree with the parties, and treat each list such that the recited “memory check” must detect at least one of the alternatives. We also note our agreement with Patent Owner that “the list *does not include* the final limitation ‘wherein when an alert of suspicious behavior . . . for the invoked function.’” Prelim. Resp. 24–25 (alterations by Patent Owner). This is consistent with the placement of the word “and” in each list to introduce the last item of the respective list, and is reflected in our paragraphing when reproducing independent claim 1 above. *See* Ex. 1001, 13:56, 14:59, 16:21. Petitioner appears also to agree because Petitioner excludes that final limitation when quoting the list recited in independent claims 1 and 9. *See* Pet. 12.

3. *Other Terms*

Neither party advocates for construction of any other term other than that they be accorded their plain and ordinary meaning. *See* Prelim. Resp. 25–26 (Patent Owner noting that Petitioner’s codefendants in related district court litigation have argued for construction of certain terms and that other terms are indefinite, but advocating that all “should be accorded their

plain and ordinary meaning”). At this juncture, we do not find it necessary to construe any additional terms for purposes of this Decision. *See Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co.*, 868 F.3d 1013, 1017 (Fed. Cir. 2017) (“[W]e need only construe terms ‘that are in controversy, and only to the extent necessary to resolve the controversy.’” (quoting *Vivid Techs., Inc. v. Am. Sci. & Eng’g, Inc.*, 200 F.3d 795, 803 (Fed. Cir. 1999))).

D. Overview of the Prior Art

1. Shukla

Shukla describes “a network accessible computer system with [a] plurality of operating systems and applications [that] is protected using a sandbox against attacks that exploit application vulnerabilities.” Ex. 1004 ¶ 32. In particular, “[t]he application sandbox prevents attacks, [and] detects and removes malware based on [] elastic or adaptive limits on application behavior.” *Id.*

Shukla explains that a “[s]andbox for an application, or a module inside the application, is a collection of rules that enforce the limits on its actions.” *Id.* The sandbox operates by “establishing boundaries for application behavior that the application must not exceed.” *Id.* Such application behavior “includes actions such as file system access, network access, registry access, data transfer, code execution, system monitoring etc.” *Id.* Thus, “[g]iven a set of rules, the sandbox can ensure that the application does not exceed the bounds as specified by the rule set.” *Id.*

Figure 2 of Shukla is reproduced below.

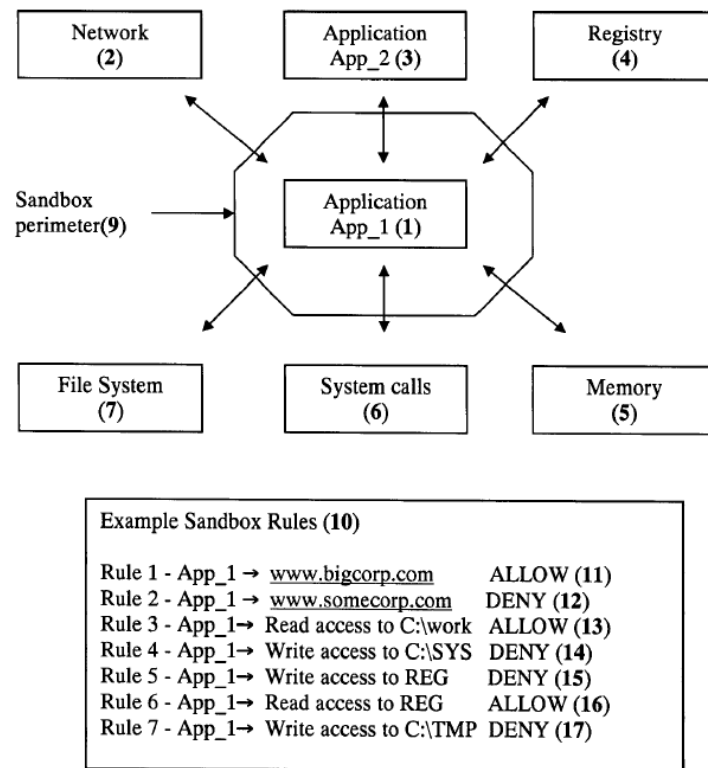


Figure 2

Figure 2 illustrates construction of sandbox 9 and sandbox rules 10 to allow 11, 13 or deny 12, 14 actions by or towards application 1. *Id.* ¶ 20. The sandbox parameters are defined “by actions [and] objects that can detect and block potential adverse effects resulting from an attack or malware,” without interfering with normal execution of application 1. *Id.* ¶ 34. Examples of such actions and objects include registry entries 4, file system access 7, network connections 2, and system calls 6. *Id.* Shukla explains that sandbox 9 is constructed by loading a module or executable code into application 1, and, if necessary, loading another component into the kernel. *Id.* ¶ 35. In this way, the sandbox is able to control interaction of the sandbox with other system components 2–7. *Id.*

According to Shukla, implementation of the sandbox inside the kernel is “a little different” from application-layer implementation because it is

possible to load driver modules inside the kernel that do not appear as a separate application. *Id.* ¶ 42. Because it is not possible to inject a module into a driver and create a sandbox, Shukla instead describes creation of a sandbox for a driver in the kernel “by scanning the memory space of the driver and hooking into the API calls made by the driver.” *Id.*

Shukla recognizes that the presence of malicious components can “either change the behavior profile of the application or prevent the sandbox [from] functioning properly.” *Id.* ¶ 60. Accordingly, Shukla describes checking applications and the kernel for malicious components “before the sandbox is enabled.” *Id.* This involves two steps: (1) “scan[ning] the application image on disk and then in memory for any known malware signatures”; and (2) searching for “behavior signatures” during the scan that “can either reveal malicious intent or hint at potentially malicious nature.” *Id.* ¶¶ 61–62.

2. Dalcher

Dalcher describes an interface monitoring method that includes “monitoring a first interface, monitoring a second interface, and taking a policy action if the second interface is not executed before the first interface.” Ex. 1005 ¶ 15. Monitoring the second interface “may include walking a call stack associated with the first interface” and “a program context for calling code associated with the second interface may be identified and acted upon.” *Id.*

Figure 2 of Dalcher is reproduced below.

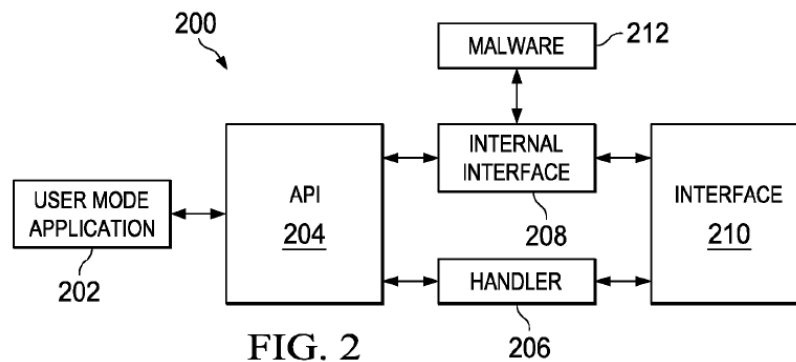


Figure 2 is a block diagram of a host environment. *Id.* ¶ 5. User mode application 202 calls “an upper-level, user mode I/O handling API 204 to write to a file.” *Id.* ¶ 42. The API “can be monitored (e.g., by hooking or host-supported monitoring) and a callback may be sent to a handler 206.” *Id.* “Handler 206 can store the event context (e.g., the name of API 204, a process identifier, etc.) and return execution to API 204, which may then call a lower-level, internal interface 208,” which may in turn call low-level interface 210. *Id.* Because “malware 212 may attempt to bypass monitored API 204 and directly call interface 208 to evade detection,” handler 206 monitors and receives a callback from interface 210 so that handler 206 “can then analyze the event context from the callback to determine if interface 210 should have been invoked by an upper-level interface, such as API 204.” *Id.* ¶¶ 42, 44.

Figure 4 of Dalcher is reproduced below.

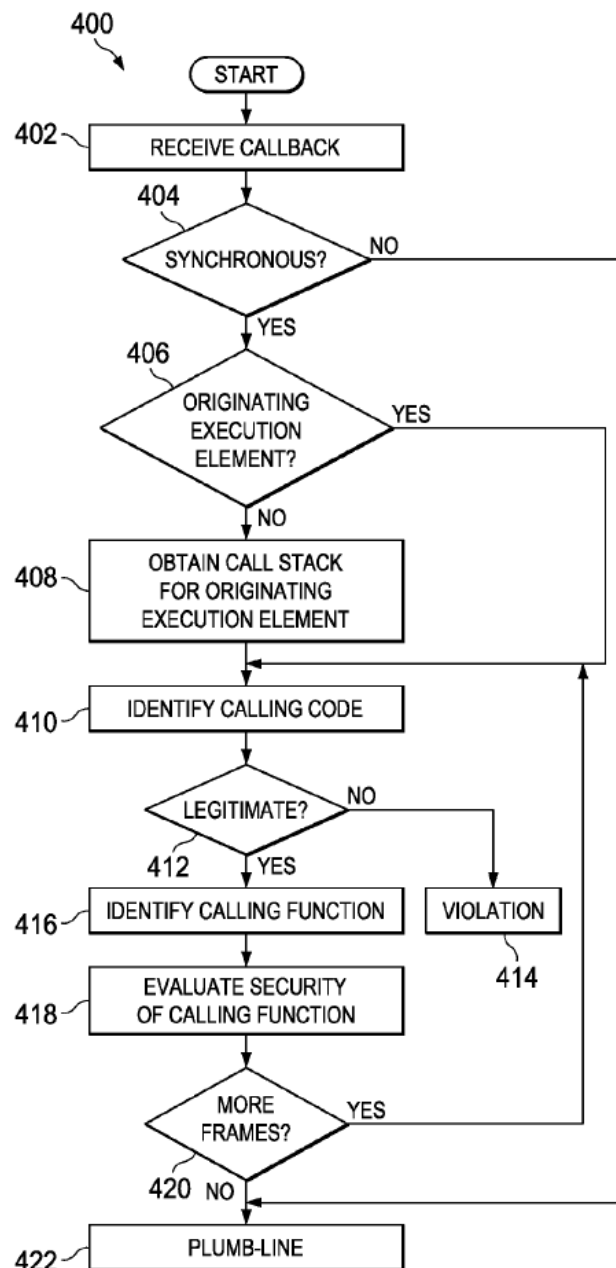


FIG. 4

Figure 4 is a flowchart that illustrates potential operations that may associated with indirect interfacing monitoring within the host environment. *Id.* ¶¶ 7, 48. When a callback is received at step 402, such as from a low-level file interface, a determination is made at step 404 whether the callback is “synchronous,” meaning that the “the original operation is held,” as

opposed to an asynchronous callback in which “the original operation is allowed to continue.” *Id.* ¶¶ 29, 48. “If the callback is determined to be synchronous at 404, but not within the originating (non-arbitrary) execution element (i.e., the originating execution element is held but is not the execution element of the callback) at 406, then access to the originating execution element’s memory can be obtained at the call stack located at 408.” *Id.* ¶ 48. “If the callback is determined to be synchronous at 404 and within the originating execution element at 406, or if the call stack is located at 408, then the call stack may be walked (e.g., using frame pointers) to identify each call frame.” *Id.* For each call frame of the call stack, “a pointer to calling code (i.e., an instruction or code that caused the call frame to be pushed onto the call stack) can be identified at 410.” *Id.*

The calling code is evaluated by analyzing the “owner of th[e] memory” at step 412. *Id.* ¶ 49. Such an evaluation “may include a comparison of the owner of the executable memory pointed to by a call frame against a discrete list of legitimate processes and modules predetermined to be associated with an interface.” *Id.* ¶ 57. Alternatively, the evaluation “may include a tracking history of the referenced memory regions, such as with behavioral detection.” *Id.* If the memory owner “is known or suspected to be malicious, then the operation may be identified as a potential security violation at 414.” *Id.* The program context may be determined at step 416, and functions identified at step 415 can be evaluated for various purposes “including assembling a collation of events for behavioral analysis, forensics, and execution profiling.” *Id.* ¶¶ 58, 62.

3. Tosa

Tosa “relates to systems and methods for protecting computer systems from malware.” Ex. 1006, 1:7–8. Figure 3 of Tosa is reproduced below.

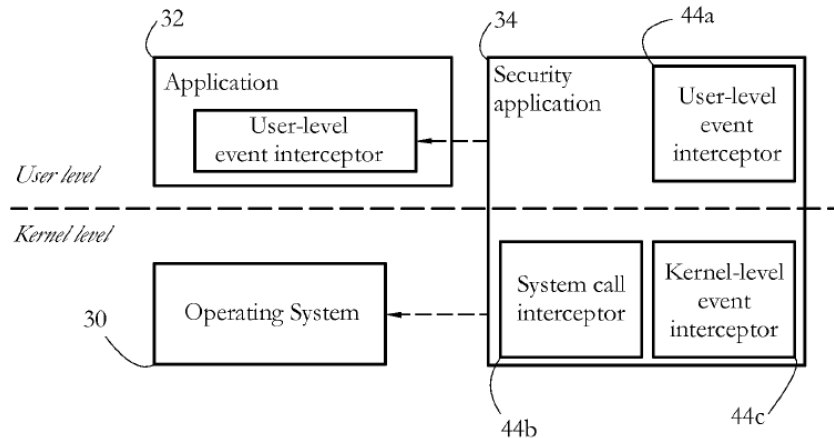


FIG. 3

Figure 3 illustrates a hierarchy of software objects executing on a host system. *Id.* at 5:4–6. Security application 34 includes user-level event-interceptor 44a having “a hooking module installed by application 34 within an evaluated application, such as application 32.” *Id.* at 5:30–35. When a process executes an instruction calling for some system functionality, such as allocating a section of memory or writing something to a disk file, “the respective instruction calls a user-mode API such as kernel32.dll or ntdll.dll.” *Id.* at 6:4–8. “Such interceptions may be achieved by any method known in the art, such as hooking[, which] is a generic term used in the art for a method of intercepting function calls, or messages, or events passed between software components.” *Id.* at 6:11–14. “[A]pplication 34 may install hooks into certain functions of kernel32.dll or ntdll.dll, to instruct the respective functions to redirect execution to event interceptor 44a,” which

may detect that the process “is attempting to perform a certain action, identified according to the hooked function.” *Id.* at 6:19–25.

4. *Kc*

Kc describes “techniques for detecting and preventing malicious code execution within a computer system.” Ex. 1007, 1:15–17. *Kc* recognizes that a “large class of successful attacks on software systems involves some form of injection of executable code via data I/O channels.” *Id.* at 2:14–17. “However, given the advent of defense techniques that are capable of detecting code executing in memory normally reserved for data, attackers have sought to redirect program control not just to injected code, but rather to existing routines such as system and execve in the standard library” in a “return-into-libc” attack, where “libc” refers to the standard C library of functions. *Id.* at 2:1–7, 2:21–28, 4:7–19. A “return-to-libc” attack thus involves overwriting code pointers to point to the address of libc functions, “typically wrapper functions that provide an indirection to system-call functions.” *Id.* at 4:12–16.

To prevent such attacks, a “libc function interception mechanism maintains an alternative wrapper function for each of the relevant standard libc routines, intercepts a call from an application to one or more libc routines and redirects the call into the corresponding alternative wrapper function.” *Id.* at 4:42–46. “A stack walk is performed along a dynamic chain of each alternative wrapper function, so as to validate call site memory addresses and target site memory addresses against start limits and end limits, respectively, for libc routine[s] associated with the alternative wrapper function.” *Id.* at 4:46–51.

Figure 3 of Kc is reproduced below.

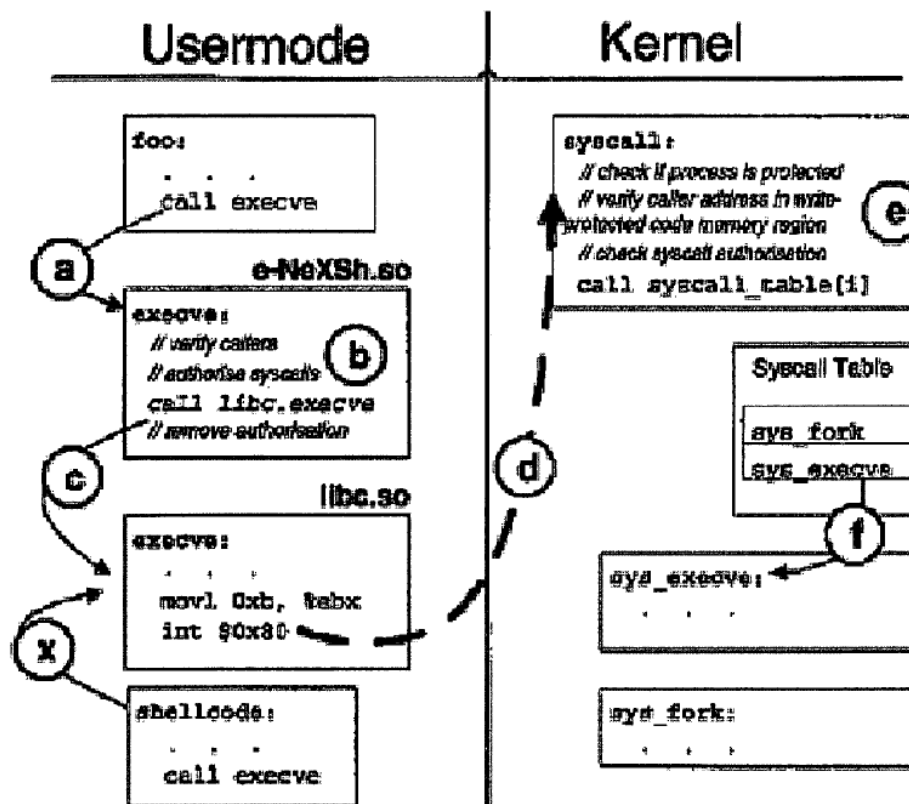


FIG. 3

Figure 3 “illustrates a successful, legitimate invocation of a system call, and an unsuccessful return-into-libc attempt by malicious code.” *Id.* at 5:36–38. “The sequence (abcdef) indicates a successful system-call invocation by ‘valid’ function ‘foo,’ being routed through the new execve wrapper function in e-NeXSH.so and the original libc wrapper function execve in order.” *Id.* at 9:25–29. Step “b” represents operations of the alternative wrapper function, which begin with the stack trace and caller-callee verification, and end with informing the kernel to authorize invocation of sys_execve before the original libc function is called. *Id.* at 9:29–33. At

step “d,” a trap instruction is executed and a system-call signal handler in the kernel is triggered. *Id.* at 9:33–39. At step “e,” the kernel performs its own checks, which include a “read-only code” check “to verify that the virtual memory address of the calling code is in read-only memory, and that the currently requested system call is authorized.” *Id.*

Thus, in this example, “[t]he sequence (xde) indicates an unsuccessful return-into-libc attempt by attack code in shellcode to invoke the same system call.” *Id.* at 9:39–41. Although the attack passes the “read-only code” check at step “e,” the attack fails the authorization check in the system call routine because the flow of control did not pass through the authorizing call in the alternative wrapper function. *Id.* at 9:41–44.

E. Challenges over Shukla and Dalcher

Petitioner challenges claims 1, 3, 9, 11, and 17 over Shukla and Dalcher; and challenges claims 4 and 12 over Shukla, Dalcher, and Tosa. Pet. 14–44.

1. Independent Claim 1

In addressing the computer-implemented method recited in independent claim 1, Petitioner proposes to incorporate Dalcher’s stack walk into Shukla’s sandbox, and contends that the resulting combination meets all limitations recited in the claim. *See* Pet. 16–35. In particular, Petitioner relies on Shukla alone for the limitations requiring monitoring of a memory space and invoking one of a plurality of functions after receiving a call from an application programming instance. *Id.* at 16–21. Patent Owner does not dispute that Shukla discloses such limitations.

With respect to the requirement that stack walk processing be executed “upon invocation of one of the plurality of functions in the monitored memory space,” Petitioner observes that “Shukla also teaches ‘the stack is analyzed to trace the module that originated the API function call,’” but concedes that “Shukla does not state whether the stack analysis includes stack walk processing as claimed by the ’591 Patent.” *Id.* at 21–22 (quoting Ex. 1004 ¶ 58). Petitioner accordingly relies on Dalcher’s disclosure of stack walking, as illustrated in Figure 4 of Dalcher, reproduced above, characterizing Dalcher’s teaching as “executing stack walk processing upon the invocation of a monitored interface function.” *Id.* at 24 (citing Ex. 1003 ¶¶ 104–106).

Petitioner reasons that, based on such a teaching, it would have been obvious for a person of ordinary skill in the art “to configure Shukla’s injected module to execute stack walk processing upon invocation of an API function in the monitored application memory space.” *Id.* at 24–25 (citing Ex. 1003 ¶ 107). As an example, Petitioner contends that a skilled artisan “would have understood that Dalcher’s stack walk processing would have been advantageously performed as part of the injected module’s ‘post processing of the results returned by the target function,’” as described by Shukla. *Id.* at 25 (quoting Ex. 1004 ¶ 54; citing Ex. 1003 ¶ 107). Relying on Dr. Lee’s testimony that “[i]t was well-known for over a decade prior to the ’591 Patent that ‘[t]he call stack of a program execution can be a very good information source for intrusion detection,’” Petitioner reasons that a person of ordinary skill in the art “would have been motivated to apply the teachings of Dalcher to enable Shukla’s injected module to perform Dalcher’s stack walk processing upon invocation of one of Shukla’s targeted

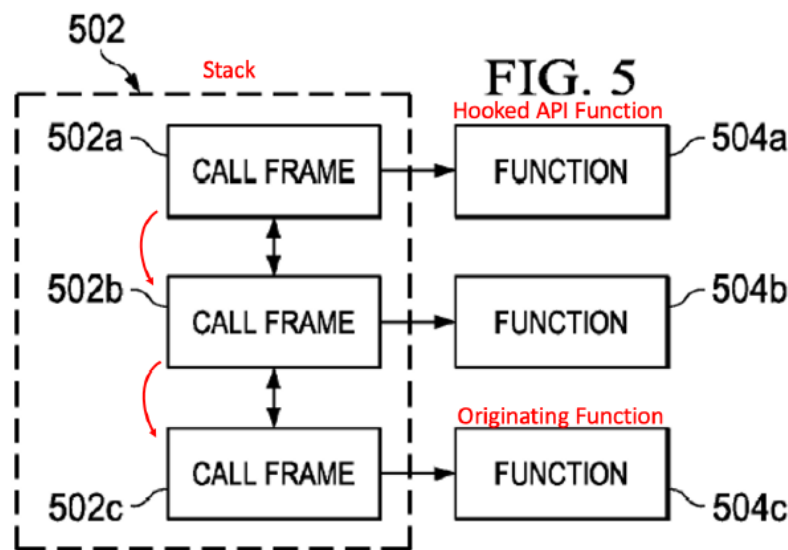
API functions to predictably improve the module's ability to detect and prevent malware intrusions to improve similar monitoring applications in the same way.” Ex. 1003 ¶ 108 (quoting Ex. 1025); Pet. 25.

Patent Owner disputes Petitioner's reasoning “because a [person having ordinary skill in the art] would never have been motivated to combine the teachings of two fundamentally different monitoring methods where doing so would serve no purpose.” Prelim. Resp. 37. According to Patent Owner, and supported by Dr. Medvidovic's testimony, a person of ordinary skill in the art “would understand that Shukla's system would only be capable of processing ‘the results returned by the target function’ for call frames that have completed execution and are no longer on the call stack.” *Id.* at 38 (citing Ex. 1004 ¶ 54; Ex. 2001 ¶ 101). Because “[t]he post-processing step would relate to only these executed and completed calls,” Patent Owner reasons that “there would be no need for gathering information or Dalcher's stack walk,” and “Dalcher's stack walk serves no practical purpose when shoehorned into Shukla's ‘post processing’ step.” *Id.* (citing Ex. 2001 ¶ 101). Patent Owner adds that “[i]mplementation of Dalcher's stack walk within Shukla's sandbox would be superfluous because Shukla's sandbox will ultimately allow execution and apply its own rule set to allow or ‘block potential adverse *effects*’—regardless of any information returned by the alleged addition of Dalcher's stack walk.” *Id.* at 38–39 (quoting Ex. 1004 ¶ 34 (emphasis by Patent Owner), citing Ex. 2001 ¶¶ 103, 105).

Both parties support their respective positions with testimony by the respective experts. The preliminary record before us presents insufficient basis to discount the testimony of either expert, and the record would benefit

from cross-examination of those declarants. Mindful of the standard we apply at this stage, we determine that Petitioner makes a sufficient showing with respect to the “executing stack walk processing” limitation.

Independent claim 1 also recites performing a memory check for a plurality of stack entries identified during the stack walk processing to detect suspicious behavior “during the executing of the stack walk processing before an address of an originating caller function is reached.” Ex. 1001, 13:41–45. Petitioner contends that this is taught by Dalcher, and that it would have been obvious for a person of ordinary skill in the art to configure Shukla’s sandbox module in accordance with that teaching. Pet. 30 (citing Ex. 1003 ¶ 117). In advancing such an argument, Petitioner observes that “Dalcher includes an exemplary call stack with a plurality of stack entries (call frames 502a-502c) in Figure 5,” and provides the following annotated version of that drawing. Pet. 27.



Dalcher’s Figure 5, reproduced above with Petitioner’s annotations, illustrates call-frame analysis with stack 502, which includes call frames

502a–502c that “store information about active functions that have been called but have not yet completed execution by returning.” Ex. 1005 ¶ 63.

Petitioner asserts that a person of ordinary skill in the art “would have understood that when implementing Dalcher’s stack walk processing in Shukla’s injected module, the stack walk processing would be performed from the stack entry associated with the intercepted API function through any intermediate stack entries until the stack entry associated with the originating function.” Pet. 28–29. Petitioner’s annotations to Dalcher’s Figure 5 are evidently meant to identify a correspondence between such an “intercepted API function” and “originating function” with Dalcher’s stack. “Thus,” Petitioner continues, “implementing Dalcher’s stack walk process in Shukla’s injected module would require starting with the stack frame entry associated intercepted low-level user mode function and walking back through the stack frame entries until the stack frame entry associated with the originating function is reached.” *Id.* at 29. Petitioner concludes that “[i]t would have also been obvious to a [person of ordinary skill in the art] to configure Shukla’s sandbox module to perform a memory check for each stack entry before an address of an originating caller function is reached as taught by Dalcher.” *Id.* at 30 (citing Ex. 1003 ¶ 117).

Patent Owner responds that “Petitioner cannot point to a single passage or figure from Dalcher specifically [to] identify ‘*an originating caller function*’ or teach stack walking towards ‘*an address*’ for the same.” Prelim. Resp. 43. Petitioner and Dr. Lee direct our attention to the description of Dalcher’s Figure 4, particularly the statement that, “[i]f additional frames remain in the stack at 420, then stack walking continues at 410.” Pet. 28 (quoting Ex. 1005 ¶ 62); Ex. 1003 ¶ 113 (quoting same). As

Dr. Medvidovic testifies, “[t]his passage merely discloses stack walking additional frames remaining on a stack,” and “does not suggest where the stack walk began or stack walking towards any specific address.” Ex. 2001 ¶ 115. While Dr. Medvidovic’s testimony appears to be accurate, it attacks Dalcher individually rather than accounting for Petitioner’s full reasoning based on the combination with Shukla. Under the standard we apply at this stage, Petitioner’s showing is sufficient for this limitation.

For independent claim 1’s requirement that an alert be triggered when performing the memory check detects “at least one of” a list of items, Petitioner focuses on the detection of an invalid stack return address that indicates a buffer overflow attack. Pet. 31–33. Petitioner specifically observes that Figure 9 of Dalcher (not reproduced here) “illustrates potential operations that may be associated with analyzing stack entries that are monitored while a stack is unwound” to detect a buffer overflow attack. *Id.* at 32 (quoting Ex. 1005 ¶ 94). At step 904,

[i]t can then be determined whether a current address points to writeable memory 904. If so, a potential security violation may have been discovered in the form of a possible buffer overflow attack in progress. Thus this situation may be flagged as a possible buffer overflow violation at 906. The return address may also be noted for execution profiling purposes at 908 if the violation is flagged at 906 or if the return address is determined not to point to writeable memory at 904.

Ex. 1005 ¶ 94. Petitioner relies on this disclosure, with testimony by Dr. Lee discussing how it integrates with the combination with Shukla, to support its position. Pet. 32–33 (citing Ex. 1003 ¶ 123).

Patent Owner disputes the sufficiency of Petitioner’s showing by again attacking the combination of the identified Dalcher disclosure within

the combination with Shukla: “Petitioner fails to recognize that Shukla’s scan for behavior signatures and potential flagging of applications is performed as part of Shukla’s preliminary ‘integrity check’—which occurs *‘before the sandbox is enabled.’*” Prelim. Resp. 46 (quoting Ex. 1004 ¶¶ 60–62) (emphasis by Patent Owner). According to Patent Owner, an incompatibility exists because “Petitioner’s entire argument relies on incorporating Dalcher’s disclosure into Shukla’s sandboxing process,” which may require “drastic modifications.” *Id.* at 47. According to Patent Owner, a person of ordinary skill in the art “would not be motivated to make alterations to accommodate the addition of an alert in its sandboxing process—particularly where the addition of a flag or ‘violation’ within the sandboxing process would serve no purpose.” *Id.* at 47–48 (citing Ex. 2001 ¶¶ 127–128).

Again, both parties support their respective positions with testimony by their respective experts. Further development of those positions through cross-examination of those experts would be beneficial. At this stage, based on the preliminary record before us, Petitioner makes a sufficient showing with respect to this limitation.

Finally, for the claim’s requirement of “preventing execution of a payload for the invoked function from operating” when an alert is triggered, Petitioner relies on Dalcher’s teaching that “an appropriate security polity may be implemented at 316, such as generating an alert or report or **halting execution.**” Pet. 34 (quoting Ex. 1005 ¶ 47) (emphasis by Petitioner). Based on such a teaching, Petitioner reasons that it would have been obvious to a person of ordinary skill in the art “to configure Shukla’s injected module to halt execution of the invoked function in response to a security violation

being triggered thereby preventing execution of a payload (i.e., the attack payload) for the invoked function from operating.” *Id.* (citing Ex. 1003 ¶ 126).

Patent Owner contends that a person of ordinary skill in the art would not have been motivated to effect the combination of teachings that Petitioner proposes because Shukla “teaches away from ‘*preventing execution of a payload*’ ‘*when an alert of suspicious behavior is triggered.*’” Prelim. Resp. 48–49. This is so, Patent Owner says, because Shukla’s sandbox, “rather than interfere with the execution of the application, ‘detect[s] and block[s] potential ***adverse effects*** resulting from an attack or malware, but does not interfere with the normal execution of the valid application 1 and allows it to function as intended.’” *Id.* at 49 (quoting Ex. 1004 ¶ 34) (alterations by Patent Owner). But Shukla’s statements do not appear to make criticisms that would rise to the level of a teaching away. *See In re Fulton*, 391 F.3d 1195, 1201 (Fed. Cir. 2004) (a prior-art reference does not teach away from the claimed subject matter unless the prior-art reference also criticizes, discredits, or otherwise discourages the solution claimed). Patent Owner will have an opportunity during the trial to revisit and elaborate on this argument.

For the reasons set forth above, we conclude that Petitioner demonstrates a reasonable likelihood of prevailing on its challenge of independent claim 1 as unpatentable under 35 U.S.C. § 103(a) over Shukla and Dalcher.

2. Independent Claims 9 and 17

Independent claims 9 and 17 are directed respectively to a “system” and a “computer-readable storage device” that correspond generally to the “computer-implemented method” recited in independent claim 1. *Compare* Ex. 1001, 13:29–61, *with id.* at 14:29–64 *and id.* at 15:34–16:27. Except for limitations that correspond to such different statutory classes, Petitioner relies on its analysis of independent claim 1. Pet. 36–39. Patent Owner makes no separate arguments for claims 9 and 17, but instead relies on the same analysis as for claim 1. *See* Prelim. Resp. 28–50.

We have reviewed Petitioner’s analysis of independent claims 9 and 17 and conclude that Petitioner demonstrates a reasonable likelihood of prevailing on its challenge of those claims as unpatentable under 35 U.S.C. § 103(a) over Shukla and Dalcher.

3. Dependent Claims 3 and 11

Claims 3 and 11 depend respectively from independent claims 1 and 9 and recite that “executing of the stack walk processing further comprises walking a stack frame associated with the invoked function from a low level user mode function to the originating address of the invoked function.” Ex. 1001, 14:1–5, 15:4–8. For this limitation, Petitioner relies on Dr. Lee’s testimony to support its assertion that a person of ordinary skill in the art “would have known that the Ntdll.dll is a user-mode linked library containing functions that directly invoke native APIs in the kernel space and therefore represent the lowest-level user mode functions in the Windows operating system.” Pet. 35 (citing Ex. 1003 ¶ 129). Petitioner argues that “the combination of Shukla and Dalcher teaches walking a stack frame

associated with the function from a low-level user mode function, such as a function in the Ntdll.dll Native Windows API, to the originating address of the invoked function.” *Id.* at 35–36. Patent Owner does not dispute this argument outside of its arguments directed at the underlying independent claims. Prelim. Resp. 66.

We conclude that Petitioner demonstrates a reasonable likelihood of prevailing on its challenge of dependent claims 3 and 11 as unpatentable under 35 U.S.C. § 103(a) over Shukla and Dalcher.

4. Dependent Claims 4 and 12

Claims 4 and 12 depend respectively from independent claims 1 and 9 and recited that “the monitoring further comprises monitoring functions associated with memory management in the memory space by a user mode function.” Ex. 1001, 14:6–9, 15:9–12. For this limitation, Petitioner contends that it would have been obvious to a person of ordinary skill in the art “to configure Shukla’s injected module to monitor functions associated with memory management in the memory space by a user mode function as taught by Tosa.” Pet. 44 (citing Ex. 1003 ¶ 153). “Since Shukla already teaches monitoring functions in the kernel32.dll library, there would have been a reasonable expectation of success configuring Shukla’s sandbox to specifically monitor the kernel32.dll library’s VirtualAlloc, VirtualProtect, HeapCreate, and HeapAlloc API functions as taught by Tosa.” *Id.* (citing Ex. 1003 ¶ 154). Petitioner supports its reasoning with Dr. Lee’s testimony, and, on the current record, Patent Owner does not dispute this reasoning outside of its arguments directed at the underlying independent claims. Ex. 1003 ¶¶ 153–154; Prelim. Resp. 66.

We conclude that Petitioner demonstrates a reasonable likelihood of prevailing on its challenge of dependent claims 4 and 12 as unpatentable under 35 U.S.C. § 103(a) over Shukla, Dalcher, and Tosa.

F. Challenges Based on Kc

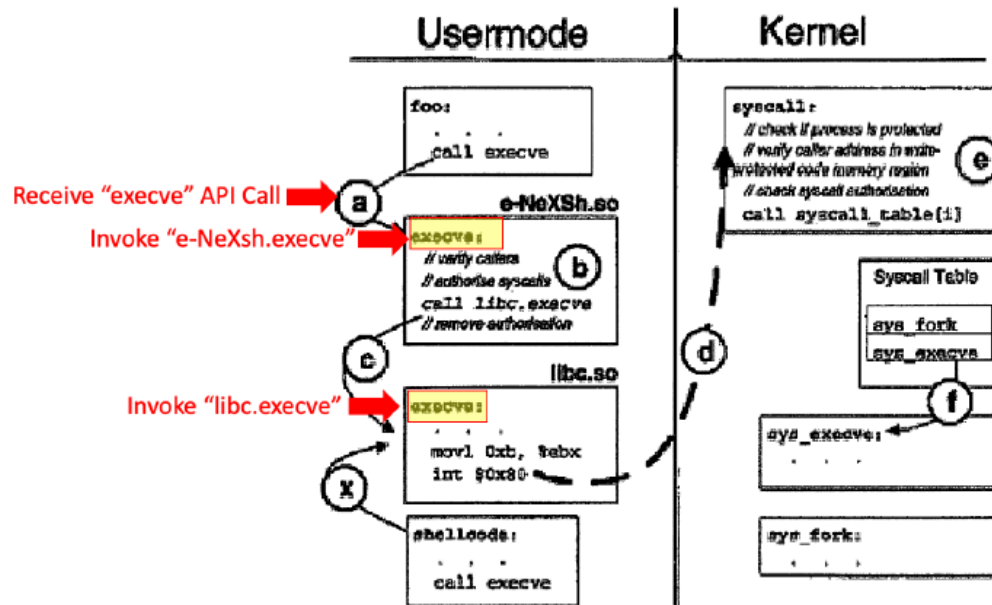
Petitioner challenges independent claims 1, 9, and 17 as anticipated by, or unpatentable for obviousness over, Kc.³ Pet. 46–61. Petitioner provides a full analysis of its challenge to independent claim 1, and relies on that analysis in challenging the other independent claims, which are directed at different statutory classes. *See id.* Patent Owner also addresses all three independent claims collectively. Prelim. Resp. 50–66. We accordingly focus on independent claim 1, with our analysis also applying generally to independent claims 9 and 17.

Patent Owner raises two principal issues in disputing the sufficiency of Petitioner’s showing. On the preliminary record, Patent Owner appears to have the stronger position.

First, in addressing the “monitoring” limitation of independent claim 1, Petitioner contends that Kc monitors a memory space for execution

³ Petitioner acknowledges that “[a] child of Kc . . . that shares the same specification was cited during prosecution of the ’591 Patent[, but] the related Kc application does not appear to have been substantively considered by the Examiner and was not relied on to reject any claims during prosecution.” Pet. 63. Patent Owner states that “[j]ust as the Kc Continuation disclosures were not invalidating at the USPTO, neither are Kc’s identical disclosures here,” but Patent Owner does not ask that we exercise our discretion under 35 U.S.C. § 325(d) to deny the Petition. Prelim. Resp. 50. We accordingly do not address whether such discretionary denial would be appropriate.

of at least one monitored libc function “of a plurality of libc functions.” Pet. 48. Petitioner thereby draws a correspondence between the recited “plurality of functions” and Kc’s libc functions. When subsequently addressing the “invoking” limitation, Petitioner provides the following annotated version of Kc’s Figure 3. *Id.* at 50.



Petitioner’s annotations of Figure 3 identify receipt of the execve call by the e-NeXSh.so shared library object, invocation of e-NeXsh.execve at step “b,” and invocation of libc.execve after step “c.” Petitioner contends that the “invoking” limitation is met because “the e-NeXSh.so shared library object receives the ‘execve’ API function call and invokes the e-NeXsh.execve [sic] wrapper function before invoking the libc.execve [sic] function.” *Id.* at 49.

Although Patent Owner contends that Petitioner’s analysis is unclear because “the petition takes two different positions as to what constitutes the invoked function,” Petitioner’s analysis appears consistent with Patent Owner’s assertion that, “[t]he original libc function . . . is not invoked until after the wrapper function has completed its stack walk and wrapper

checks.” Prelim. Resp. 55–58. We need not resolve Patent Owner’s clarity contention at this time, and we need not decide whether invocation of `libc.execve` after invocation of `e-NeXsh.execve` amounts to invocation of `libc.execve` “as a result of” receiving the original call of `execve` at step “a.” This is because Petitioner, when addressing the “executing stack walk processing” limitation, clearly contends that the limitation is met because Kc teaches performing a stack walk upon invocation of the wrapper function `e-NeXsh.execve`. Pet. 50–51.

Petitioner’s analysis appears faulty because Petitioner identifies `libc.execvc` as the claim’s recited “one of the plurality of functions” invoked in the “invoking” step, but identifies `e-NeXsh.execve` as the “one of the plurality of functions” invoked in the “executing” step. That is, as Patent Owner and Dr. Medvidovic say, “Kc does not teach executing a stack walk *‘upon the invocation of one of the plurality of functions’*” because “Kc executes a stack walk upon invoking the alternative wrapper function (`e-NeXSh.execve`) and not the original `libc` function.” Prelim. Resp. 58–59; Ex. 2001 ¶¶ 157, 159.

Second, in addressing the portion of the “performing” limitation that requires triggering “an alert of suspicious behavior,” Petitioner relies on Kc’s disclosure of a “process enabling flag.” Pet. 54. Specifically, Kc teaches that described embodiments “add a flag to determine whether the process should be subject to the `e-NeXSh` protection.” Ex. 1007, 9:65–67. “If a process does not have this flag set,” Kc adds, “its system calls are not checked to see if they are made from write-protected memory pages.” *Id.* at 9:67–10:2. Kc explains that “[h]aving the option to selectively protect applications allows one to take any program that does not need the

protection and run it unmodified, completely unaffected by the system-call interposition mechanism.” *Id.* at 10:15–18. This is advantageous because “not all programs that run on a system are subject to external attacks—the focus of the described embodiment is on server software that does need protecting,” such as “database servers that actually have to interact with possibly malicious clients on the Internet.” *Id.* at 10:18–23.

In light of Kc’s teaching of such a “process enabling flag,” “Petitioner maintains Kc either expressly or inherently discloses triggering an alert of suspicious behavior when the stack walk verification process fails.” Pet. 55. Alternatively, Petitioner contends that “this would have also been obvious to a [person having ordinary skill in the art].” *Id.* (citing Ex. 1003 ¶¶ 178–179). To support this assertion regarding obviousness, Petitioner relies on the testimony of Dr. Lee, who states that “[i]t would have been obvious to a person having ordinary skill in the art to configure Kc’s libc function interception mechanism 104 to trigger an alert of suspicious behavior when the memory check detects a return address outside the write-protected memory region reserved for the application code indicating a probable return-into-libc attack.” Ex. 1003 ¶ 179.

There are a number of difficulties with Petitioner’s position. First, as Patent Owner suggests, Kc’s “process enabling flag” “is not an alert; rather the flag is used to selectively identify applications that should be subject to Kc’s security process.” Prelim. Resp. 60 (citing Ex. 2001 ¶¶ 162–163). In addition to being supported by Dr. Medvidovic’s testimony, this is evident from Kc’s description of the flag, discussed above, and, as Patent Owner points out, the name of the flag. *See id.* Even Petitioner’s expert, Dr. Lee, testifies that “the operating system can halt the execution of malicious code

when the process enabling flag is set.” Ex. 1003 ¶ 176. Because Kc’s flag provides an option for selective protection, Patent Owner further reasons that it must be set before the security process even begins, such that “it is not an alert of suspicious behavior that is triggered by any steps within the process.” Prelim. Resp. 60–61 (citing Ex. 2001 ¶¶ 164–165).

With respect to inherency or obviousness of an alert in light of Kc’s teachings, Patent Owner observes that “Kc uses the opposite technique taught in the ’591 patent (triggering an alert of suspicious behavior) by triggering **authorization** based on a **lack** of suspicious behavior.” Prelim. Resp. 61. Supported by Dr. Medvidovic’s testimony, Patent Owner reasons that “an **alert** of suspicious behavior would have been neither inherent nor obvious.” *Id.* (citing Ex. 2001 ¶¶ 166–168). Expressed differently, Patent Owner states that, “whereas the ’591 patent will allow code (including a system call) to execute unless an alert is triggered, Kc will **not** allow a system call to execute unless authorization is given.” *Id.* at 62.

Patent Owner’s argument is persuasive: “Because Kc uses the opposite technique for authorizing system calls, Kc does not inherently disclose an alert of suspicious behavior”; and “because adding such an alert would have created redundancy and required substantive alteration of Kc’s method, it would not have been obvious to do so. *Id.* (citing Ex. 2001 ¶ 168).

For these reasons, we conclude that Petitioner does not demonstrate a reasonable likelihood of prevailing on its challenge of independent claims 1, 9, and 17 as anticipated by, or unpatentable for obviousness over, Kc. Nevertheless, should it wish to do so, Petitioner will have an opportunity during the trial to address these specific concerns.

G. Objective Indicia of Nonobviousness

Patent Owner contends that it “provid[ed] evidence regarding the long-felt unmet need for the invention in its district court complaint against Petitioner,” and that, because “Petitioner fails to consider such evidence in its Petition . . . institution should be denied.” Prelim. Resp. 67 (citing Ex. 2003, 10, 24–26).

In evaluating whether an invention would have been obvious, “[s]uch secondary considerations as commercial success, long felt but unsolved needs, failure of others, etc., might be utilized to give light to the circumstances surrounding the origin of the subject matter sought to be patented.” *Graham v. John Deere Co.*, 383 U.S. 1, 17–18 (1966). Although it is Patent Owner’s burden to introduce evidence supporting such objective indicia, *see In re Huang*, 100 F.3d 135, 139 (Fed. Cir. 1996), the ultimate burden of persuasion to prove unpatentability of the challenged claims never shifts to Patent Owner, *see Dynamic Drinkware, LLC v. National Graphics, Inc.*, 800 F.3d 1375, 1378 (Fed. Cir. 2015). Rather, objective indicia should be considered along with all of the other evidence in making an obviousness determination. *See Eurand, Inc. v. Mylan Pharm. Inc. (In re Cyclobenzaprine Hydrochloride Extended-Release Capsule Patent Litig.)*, 676 F.3d 1063, 1076–77 (Fed. Cir. 2012) (“It is to be considered as part of all the evidence, not just when the decisionmaker remains in doubt after reviewing the art.”) (citing *Stratoflex, Inc. v. Aeroquip Corp.*, 713 F.2d 1530, 1538–39 (Fed. Cir. 1983)).

We have reviewed the portions of the district court complaint that Patent Owner cites. Although Patent Owner characterizes those passages as

“providing evidence” of a long-felt need, they cite only to the ’591 patent itself. *Id.*; *see* Ex. 2003, 10, 24–26. We do not discern evidence with the kind of specificity for objective indicia of nonobviousness, notably including evidence of a nexus with the merits of the challenged claims, that would cause us to fault Petitioner for failing to address the evidence in its Petition. *See Lectrosonics, Inc. v. Zaxcom, Inc.*, IPR2018-01129, Paper 33, 31–35 (PTAB Jan. 24, 2020). We accordingly decline to deny the Petition on this basis alone. The parties will have an opportunity during the trial to develop Patent Owner’s evidence of objective indicia of nonobviousness so that it may be fully considered.

III. CONCLUSION

For the foregoing reasons, we conclude that the information presented in the Petition establishes a reasonable likelihood that Petitioner would prevail in showing that at least one of the challenged claims is unpatentable. Accordingly, we institute *inter partes* review as to all challenged claims and grounds presented in the Petition. *See SAS Inst. Inc. v. Iancu*, 138 S. Ct. 1348, 1359–60 (2018); 37 C.F.R. § 42.108(a). In light of our analysis above, we anticipate that the trial will more productively focus on the grounds that involve Shukla and Dalcher. Nevertheless, informed of our initial analysis on the preliminary record, the parties are free to develop their positions for any grounds that they choose.

IV. ORDER

In consideration of the foregoing, it is

ORDERED that, pursuant to 35 U.S.C. § 314(a), *inter partes* review is hereby instituted as to claims 1, 3, 4, 9, 11, 12, and 17 of the '591 patent on all grounds set forth in the Petition; and

FURTHER ORDERED that, pursuant to 35 U.S.C. § 314(c) and 37 C.F.R. § 42.4, notice is hereby given of the institution of a trial, commencing on the entry date of this Decision.

IPR2022-01522
Patent 10,284,591 B2

For PETITIONER:

Adam Seitz
Paul Hart
Callie Pendergrass
ERISE IP, P.A.
adam.seitz@eriseip.com
paul.hart@eriseip.com
callie.pendergrass@eriseip.com

For PATENT OWNER:

Brain Eutermoser
Russell Blythe
Mikaela Stone
KING & SPALDING LLP
beutermoser@kslaw.com
rblythe@kslaw.com
mikaela.stone@kslaw.com