UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CROWDSTRIKE, INC.,
Petitioner

v.

TAASERA LICENSING LLC,
Patent Owner

Case No. IPR2024-00027
Patent No. 7,673,137

PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 7,673,137

TABLE OF CONTENTS

I. INTRODUCTION

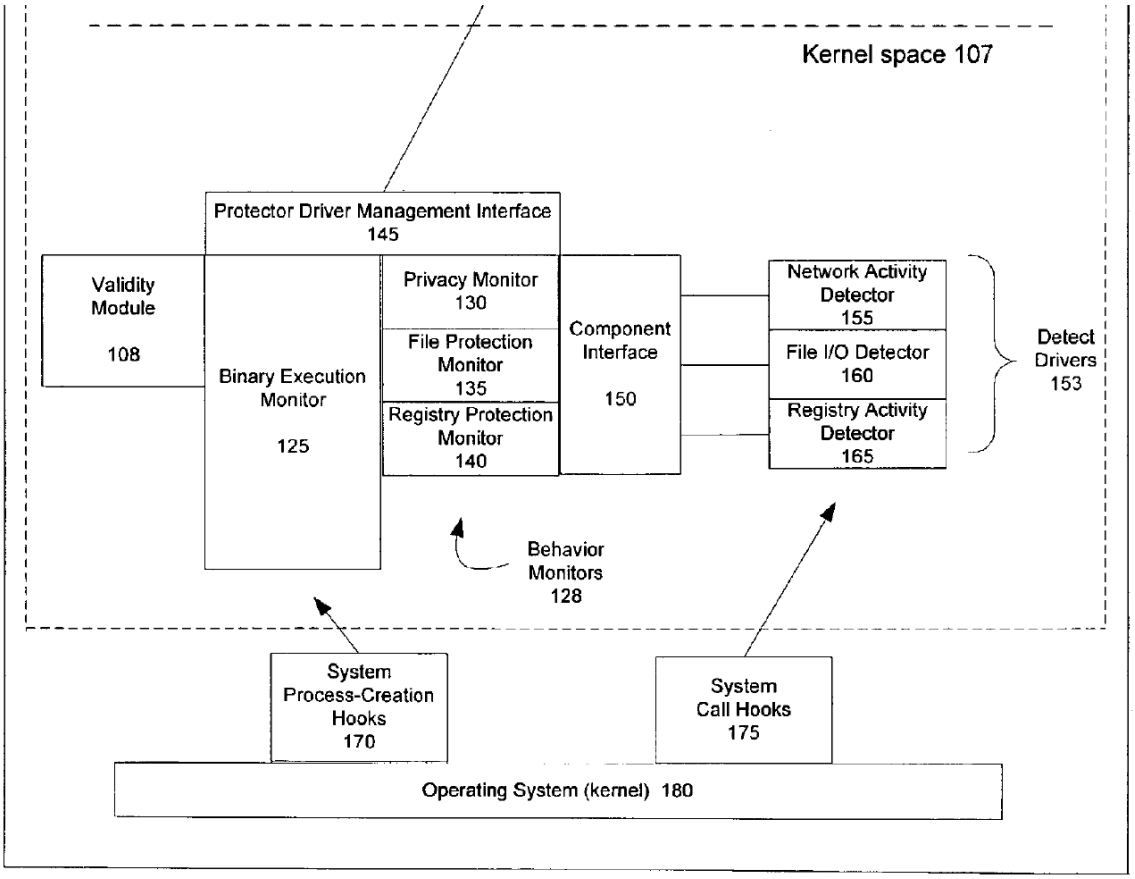
CrowdStrike, Inc. ("Petitioner") respectfully requests *inter partes* review ("IPR") of claims 1, 2, 6, 8, 9, 12, and 25 (collectively, the "Challenged Claims") of U.S. Patent No. 7,673,137 (the "'137 Patent").

II. SUMMARY OF THE '137 PATENT

A. Description of the '137 Patent

The '137 Patent proposes a system that "implement[s] a two-phased approach at the kernel level of the computing device's operating system…[to] provide[] fast and efficient security that minimizes interruptions for the user." *'137 Patent* (Ex. 1001), 3:25-28. "[T]he pre-execution process, performs a rapid validation check to determine whether the program has been approved for the computing device or network." *Id.*, 3:28-31. The pre-execution phase "occurs as a program is being loaded into memory, but before it can execute." *Id.*, 8:30-32. "[T]he binary execution monitor 125 will respond to the system process-creation hook 170 by suspending execution of the executable file…until the program can be validated." *Id.*, 8:58-62. "The binary execution monitor 125 is an in-kernel driver that monitors the loading of binary executable files and other executable libraries in real time by recognizing and validating any executable file that is being loaded." *Id.*, 6:43-47. If "the binary execution monitor 125 is able to validate the new executable, the suspended state will be released…and loading of the executable will continue[.]" *Id.*, 9:4-7.

"However, if the binary execution monitor 125 is unable to validate the executable in the pre-execution process, additional precautionary steps will have to be taken in the execution phase[.]" *Id.*, 9:7-10.



Id., Fig. 1 (excerpt).

In the execution phase, an unvalidated program "can continue to load and execute, but other execution security modules are responsible for detecting, monitoring, and responding to suspicious activities." *Id.*, 3:54-57. "[D]etect drivers 153 can monitor the non-validated program when it is executing and identify potential threats to the computing device 103 before they are executed." *Id.*, 6:64-

67. "[I]f the program is attempting to perform functions that may seriously impair the computing device 103 or the network 102, the protector system 104 may immediately terminate execution of the program." *Id.*, 10:41-45.

B. Summary of the Prosecution History

The application that issued as the '137 Patent claims priority to a provisional application filed on January 4, 2002. Because the '137 Patent claims the benefit of a filing date before March 16, 2013, Petitioner applies the pre-AIA versions of §§ 102 and 103. Petitioner also adopts January 4, 2002, as the priority date for the Challenged Claims for purposes of this proceeding.

The '137 Patent underwent several rounds of rejections during prosecution. *'137 File History* (Ex. 1002). However, the primary references considered by the examiner include USPN 7,398,532 to Barber ("Barber"), USPN. 6,128,774 to Necula ("Necula"), and USPN 2002/0184520 to Bush ("Bush"). *Id.*, 1358-1360. The examiner found that none of these references teach the step of "if the new program is not the same as the allowed program, monitoring the new program while allowing it to execute on the computing device, wherein the step of monitoring the new program while allowing it to execute is performed at the operating system kernel of the computing device." *Id*., 1358-1361.

The Examiner found that unvalidated processes in Barber "will not execute," which is contrary to the allegedly patentable requirement for "monitoring the new

program while allowing it to execute on the computing device." *Id*., 1358. Likewise, Necula taught discarding invalid code. *Id*., 1359. Finally, the examiner found that while Bush teaches a sandbox where "untrusted code is permitted to execute," Bush does so on "a virtual machine, which is separate from the operating system to process untrusted code." *Id.*, 1359-1360.

As shown below, the prior art cited in the Petition expressly teaches monitoring an untrusted program on the operating system kernel of a computing device while the program executes. *Infra.*

C. Claim Construction Under 37 C.F.R. §42.104(b)(3)

In this proceeding, claims are interpreted under the same standard applied by Article III courts. 37 C.F.R. § 42.100(b); *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312 (Fed. Cir. 2005) (*en banc*). Petitioner applies the ordinary and customary meaning of the claim terms as understood by a PHOSITA for all terms not expressly constructed below.

1. "if the new program is validated…"

Claim 6 recites the limitation "if the new program is validated, permitting the new program to continue loading and to execute in connection with the computing device." Claim 25 similarly recites "if the new program is valid, permitting the new program to execute on the computing device." Petitioner proposed construing claim 6 as "if the new program is validated, then it is not monitored while it loads and

executes in connection with the computing device" and claim 25 as "if the new program is valid, then it is not monitored while it executes on the computing device."

Consistent with the '137 Patent's goal of providing "an effective and efficient method for implementing security while minimizing the burdens and interruptions for the user," if a program is validated, it is not monitored. *'137 Patent*, 10:49-54. PO's expert in the co-pending litigation recognizes this. *Cole Declaration* (EX. 1008), ¶53 ("one of the benefits of the claimed invention is performing a validation step … to minimize unnecessary monitoring...").

Petitioner's proposal is supported by the claim language, which is structured in a conditional format such that whether monitoring happens depends on whether the program was validated (e.g., "if the new program is validated, permitting the new program to continue loading;" "if the new program is not validated, monitoring the new program while it loads").

The intrinsic record uniformly describes that programs are monitored if they are not validated. The specification states that "***the present invention*** employs a two-step process to ***validate*** software programs and ***monitor non-validated*** software programs. . . . In the first phase of the process, the protector system ***validates*** authorized programs to ensure that they have not been corrupted ***before running them. For programs that cannot be validated***, the protector system can monitor the

programs as they execute during the second phase." *'137 Patent*, 4:53-65, claims. 6, 13, 14, 24; 3:25-41, 6:64-67, 9:7-11, 9:38-41.

By contrast, validated programs are not monitored and do not need to be. *Id.*, 4:54-65, cls. 6, 13, 14, 24, 3:28-32. This is common sense in view of the '137 Patent's purpose: a program known to be valid does not need to be monitored, thus saving resources. *Id*., 10:54-59; 4:4-11; 8:63-9:7. PO's claim construction expert admits that the specification's execution module is for non-validated programs. *Cole Declaration*, ¶53.

In the district court, PO has pointed to a single passage in the specification that refers to "little… additional security monitoring" to claim that "a program may still be monitored (either minimally or not at all) as it continues loading and executes" (*Taasera's Claim Construction Brief* (Ex. 1009), 5), but this does not outweigh the plain claim language and the other intrinsic evidence.

2. "pre-execution monitor"

Claim 1 recites "a pre-execution **module**" and "the pre-execution **monitor**." In co-pending district court litigation, the parties have agreed that "pre-execution monitor" refers to the "pre-execution module."

3. "an execution module"

Claim 1 recites "an execution module coupled to the detection module and operable for monitoring, at the operating system kernel of the computing device, the

program in response to the trigger intercepted by the detection module." In the co-pending district court litigation, both parties have agreed that this limitation should be subject to 112(6) and the function is "monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module." *Joint Claim Construction Brief* (Ex. 1010), 5. For purposes of this proceeding,[1] Petitioner proposes that the corresponding structure for this limitation is "if the program was not validated, then monitor the non-validated program in response to triggers while the program is executing" for the same reasons as discussed above with reference to the "if the new program is validated" phrase. *See* Section II.C.1.

D. Level of Skill of a Person Having Ordinary Skill in the Art

As of January 4, 2002, a person having ordinary skill in the art (PHOSITA) would be a person having a bachelor's degree in computer science, or a related field, or an equivalent technical degree or equivalent work experience, and an additional one to two years of education or experience in the field of computer security. More education can supplement practical experience and vice versa. *Declaration* (Ex.

[1] In the co-pending litigation, Petitioner proposes this limitation is indefinite for insufficient corresponding structure. Petitioner also provided an alternative structure in the event the Court did not find the limitation indefinite, which Petitioner applies here.

1003), 44.

III. REQUIREMENTS UNDER 37 C.F.R. § 42.104

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

Petitioner certifies the '137 Patent is eligible for IPR.

B. Identification of Challenge Under 37 C.F.R. § 42.104(b)

Petitioner requests the Challenged Claims be found unpatentable on the following grounds:

Proposed Grounds of Unpatentability	Exhibits
Ground 1: Claims 6, 8, 9, 12, and 25 are **obvious** under § 103(a) over *ChakraVyuha (CV): A Sandbox Operating System Environment for Controlled Execution of Alien Code* to Dan ("Dan") in view of USPN 2002/0099952 to Lambert ("Lambert").	1005-1006
Ground 2: Claims 1 and 2 are **obvious** under § 103(a) over Lambert in view of Dan in further view of USPN 7,085,928 to Schmid ("Schmid").	1005-1007

IV. SUMMARY OF THE PRIOR ART

A. Dan

Dan discloses that "[t]he rapid growth in connectivity and sharing of information via the World-Wide Web has dramatically increased the vulnerability of client machines to alien codes." *Dan* (Ex. 1005), 1. "The alien code can silently obtain unauthorized access to all the system resources (i.e., logical resources such as files, network ports) that can be accessed by the invoking user, and "[i]t may also

propagate itself, attempt illegal break-ins or maliciously alter files in the client system[.]" *Id.*

To combat these issues, Dan proposes a system that "combines certification of alien code together with explicit granting of privileges for execution." *Id.*, 2. Dan's system "consists of one or more code production systems, certification agencies, servers and clients." *Id*., 5.

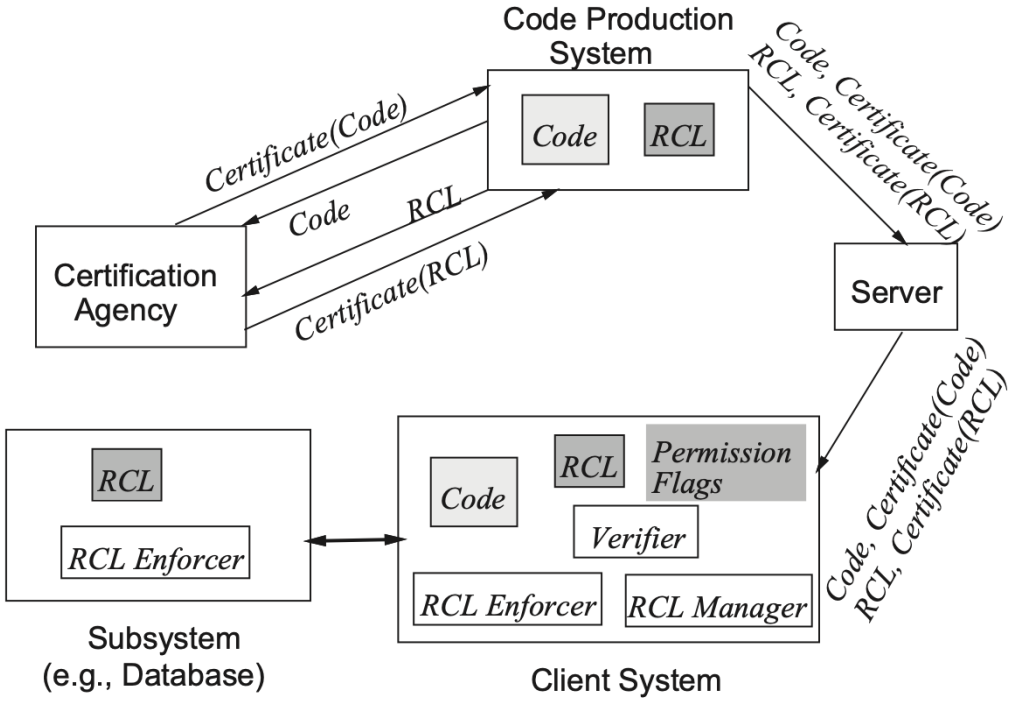


Figure 1: A. Block diagram of the proposed system.

Id.

"The certification agency issues a certificate for the code identifying its source and a certificate for the RCL of that code, where the resource capability list (RCL) is "the set of permissions and resource access privileges required by the code." *Id.*,

4-5. "The RCL enforcer module at the client ensures that the permissions and resource consumption limits specified in the RCL are not violated." *Id.*, 9. "The RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system." *Id.*, 12. "This monitoring requires that the RCL enforcer be invoked for each system call made by an application," which is implemented by modifying the operating system's call interface. *Id.,* 12, 14-15. *Dan* was not cited or considered during the prosecution of the '137 Patent.

1. Dan Is Prior Art

The determination of whether a document is a "printed publication" under 35 U.S.C. § 102(b) "involves a case-by-case inquiry into the facts and circumstances surrounding the reference's disclosure to members of the public." *In re Klopfenstein*, 380 F.3d 1345, 1350 (Fed. Cir. 2004). "A reference will be considered publicly accessible if it was 'disseminated or otherwise made available to the extent that persons interested and ordinarily skilled in the subject matter or art exercising reasonable diligence, can locate it.'" *Blue Calypso, LLC v. Groupon, Inc.*, 815 F.3d 1331, 1348.

The following facts show Dan was a prior art printed publication as of July 22, 1997:

- Dan indicates that it was an "IBM Research Report" that "has been submitted for publication outside of IBM[.]" *Dan*, 1. These indica suggest that Dan was to be made public.
- June Munford is a Librarian having a Master of Library and Information Science and over ten years of experience in the library/information science field, including experience as a Library Director. *Munford Declaration* (Ex. 1004), ¶¶2-4. Ms. Munford located a copy of Dan in the Carnegie Mellon University Library ("the Library"). *Id.,* ¶9.
- The Library's MARC record for Dan shows it was acquired and cataloged by the Library on July 3, 1997. *Id.,* ¶11, ¶14.
- A date stamp on the final page of the Library's copy of Dan indicates that it was publicly accessible as of July 22, 1997. *Id.,* ¶12, ¶14.
- In 1997, the Library's catalog records could be searched via "an online catalog that allows users to search by words/phrases and review lists of the library's authors, titles, subjects, series, keywords, or call numbers." *Id.*, ¶20. The Library "implemented Sirsi's Unicorn and WebCat software to create a detailed inventory of the library's holdings and present users with an interactive library catalog to search this inventory." *Id.*, ¶26. Dan was sufficiently indexed to allow any library catalog user to locate Dan via common search engine practices. *Id.*

- Members of the interested public exercising reasonable diligence could have located the Dan reference as of July 22, 1997. *Id.*

Thus, a preponderance of the evidence shows Dan is prior art under 35 U.S.C. §102(b).

2. Dan Is Analogous Art

Because *Dan*, like the '137 Patent, discloses systems and methods for analyzing and resolving potential security threats to a computing device, *Dan* is in the same field of endeavor as the '137 Patent. *Compare Dan*, 21 (discussing alien code and malicious propagation of alien code), 5, 1 (discussing denial of service attacks), 2 (discussing active content viruses), 12 (discussing monitoring system calls), 14-15, Figure 1 *with* '*137 Patent*, Abstract, 3:5-14, 6:21-24, 7:15-21, 8:20-27; *Declaration*, ¶78. *Dan* is also directed at solving similar problems, such as detecting security threats to a computing device. *Compare Dan*, 2, 5-6, 12, 14-15, 21, Figure 1 *with* '*137 Patent*, 3:5-14, 3:25-35, 4:55-60, 10:49-54; *Declaration*, ¶79.

B. <u>Lambert</u>

Lambert discloses that "[c]omputer users, and particularly business users, typically rely on a set of programs that are known and trusted." *Lambert* (Ex. 1006), [0003]. "However, in most enterprise environments, particularly in larger environments in which some flexibility is important with respect to what various employees need to do with computers to perform their jobs, users are able to run

unknown and/or untrusted code, that is, programs outside of the supported set." *Id*. "[S]ome of the untrusted code may be malicious code…that intentionally misrepresents itself to trick users into running it, typically inflicting damage, corrupting data or otherwise causing mischief to the computers on which the malicious code runs." *Id.*, [0004].

To solve this problem, Lambert describes "[a] system and method that automatically, transparently and securely controls software execution by identifying and classifying software, and locating a rule and associated security level for executing executable software." *Id.*, Abstract. Lambert discloses a pre-execution phase where a software file is identified and classified as a trusted application that is allowed to execute without restrictions, a potentially untrusted application that is allowed to execute with restrictions, or a "bad" application that is not permitted to execute on the computing device. *Id.*, [0056], Fig. 5A. When an untrusted program is executing in restricted mode, a security mechanism ensures that the program does not violate the restrictions defined in a restricted token associated with the program. *Id.*, [0043], Fig. 3.

1. Lambert Is Prior Art

Lambert was filed on June 8, 2001, and qualifies as prior art to the '137 Patent under 35 U.S.C. §102(e). Lambert was not cited or considered during the prosecution of the '137 Patent.

Lambert also claims priority to a provisional application filed on July 24, 2000. If PO attempts to antedate Lambert's June 2001 priority date, Petitioner will establish that Lambert qualifies as §102(e) art as of the July 24, 2000 provisional filing date. Since PO would first be required to establish that the Challenged Claims are entitled to a priority date earlier than Lambert's June 8, 2001 filing date, Petitioner is not required to show that Lambert receives the benefit of the July 24, 2000 provisional date until PO has made this showing. *In re Magnum Oil Tools Int'l, Ltd.*, 829 F.3d 1364, 1375–76 (Fed. Cir. 2016).

2. Lambert Is Analogous Art

Because Lambert, like the '137 Patent, discloses systems and methods for analyzing and resolving potential security threats to a computing device, *Lambert* is in the same field of endeavor as the '137 Patent. *Compare Lambert*, Abstract, [0007], [0009], [0013]-[0014], [0041], [0056], Fig. 5A *with* '137 Patent, Abstract, 3:5-14, 6:21-24, 7:15-21, 8:20-27; *Declaration*, ¶90. *Lambert* is also directed at solving similar problems, such as detecting potential security threats to a computing device. *Compare Lambert*, Abstract, [0004]-[0005] (discussing detecting a virus and unknown code), [0008] (discussing automatically controlling software execution), [0011] (discussing automatically applying a policy), [0014] (discussing security is implemented without requiring user forethought or action), *with* '137 Patent, 3:5-14, 3:25-35, 4:55-60, 10:49-54; *Declaration*, ¶91.

C. Schmid

"While most well-behaved Windows applications make system calls through the Win32 API, it is also possible to bypass this interface by making calls directly to kernel services." *Schmid* (Ex. 1007), 5:29-32. "A malicious program…may attempt to bypass a wrapper implemented between Windows and the Win32 subsystem by making direct calls to operating system objects, rather than through the Win32 API." *Id.*, 5:33-37.

To solve this problem, Schmid discloses a "kernel-level wrapping approach to executable control" where a device driver is "installed in an I/O subsystem, and may be used to intercept system service requests." *Id.*, 5:38-48. The device driver is "dynamically loaded into a kernel, or may be loaded as part of a boot sequence" and "modifies an entry in a table consulted by a dispatcher to handle an interrupt instruction." *Id.*, 6:6-11. "In the preferred Windows NT embodiment, the modified entry may cause a dispatcher to call an inserted function[.]" *Id.*, 6:11-15. "When a substitute function is called by a dispatcher, the function determines whether to allow or block process creation." *Id.*, 6:22-24.

1. Schmid Is Prior Art

Schmid was filed on March 30, 2001, and qualifies as prior art to the '137 Patent under 35 U.S.C. §102(e). Schmid was not cited or considered during the prosecution of the '137 Patent. Schmid also claims priority to a provisional

application filed on March 31, 2000. If PO attempts to antedate Schmid's March 2001 priority date, Petitioner will establish that Schmid qualifies as §102(e) art as of the March 31, 2000 provisional filing date.

2. Schmid Is Analogous Art

Because Schmid, like the '137 Patent, discloses systems and methods for analyzing and resolving potential security threats to a computing device, Schmid is in the same field of endeavor as the '137 Patent. *Compare Schmid*, Abstract, 2:20-23, 2:31-33, 3:21-22, 2:50-54, Fig. 1 *with '137 Patent*, Abstract, 3:5-14, 6:21-24,7:15-21, 8:20-27; *Declaration*, ¶95. Schmid is also directed at solving similar problems, such as detecting potential security threats to a computing device. *Compare* Schmid, Abstract (discussing preventing software from executing without prior approval), 2:20-23 (discussing addressing emerging dangers and unknown attacks), 2:26-28 (discussing a kernel intercepting process creation requests), 2:50-54 (discussing providing administrators with defense against hostile or unknown code), *with* '137 Patent, 3:5-14, 3:25-35, 4:55-60, 10:49-54; *Declaration*, ¶96.

V. THE CHALLENGED CLAIMS ARE UNPATENTABLE

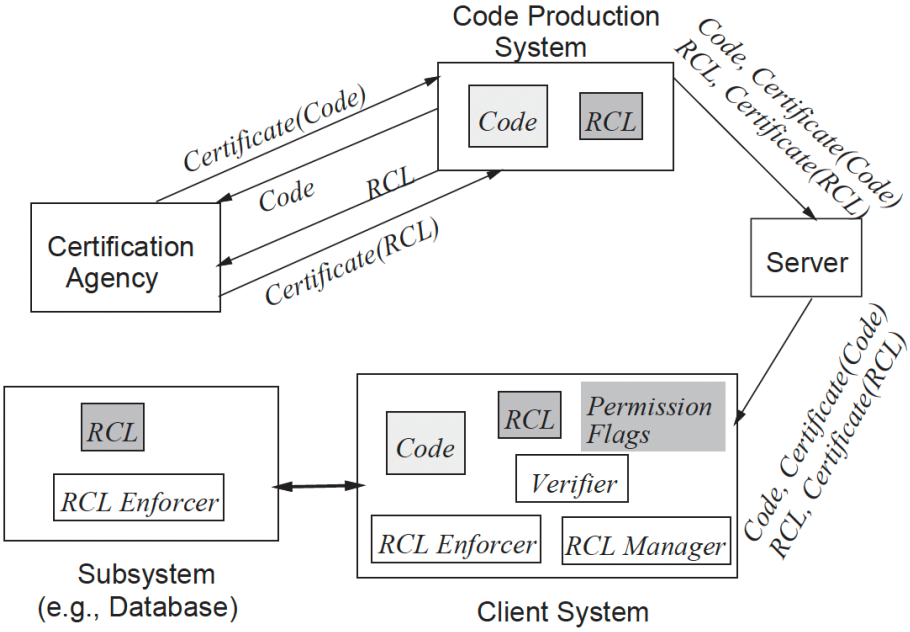
A. Ground 1: The Combination of Dan and Lambert Renders Claims 6, 8, 9, 12, and 25 Obvious

1. Claim 6

6[Pre]. A computer implemented method for implementing security for a computing device comprising the steps of:

Dan teaches a system including a client system having a sandbox environment

installed on it:



Dan, 5; *see also, id.* ("[T]he RCL may be the default RCL associated with the **sandbox environment** in the client system.").[2]

The client system includes an "operating system kernel," logical resources, such as "files [and] network ports," and physical resources, such as "disk, physical memory, [and] CPU." *Id.,* 5, 7, Abstract. A PHOSITA would have understood that an operating system kernel and logical and physical resources are all well-known components of computing devices. *Declaration*, ¶¶98-99. Therefore, a PHOSITA would understand that Dan's client system is *a computing device*.[3] *Id.*

The combination of Dan and Lambert embodies *a computer implemented*

[2] All emphases added by Petitioner unless stated otherwise.

[3] Claim language emphasized for clarity.

method for implementing security for a computing device, as set forth below.

[6(a)] interrupting the loading of a new program for operation with the computing device;

Dan teaches *a new program*. Specifically, Dan teaches a method for "Installation of Untrusted Applications" where "[t]he application and RCL are downloaded to the host (via network, diskette, CD-ROM, satellite, etc.)." *Dan*, 20. Thus, Dan's application is *new* because it is newly downloaded to the host computer. *Declaration*, ¶101. A PHOSITA would have understood that an application is a type of computer program. *Id*., ¶102-103; *see also, Dan*, Abstract ("Sharing of unknown **programs**…").

Dan describes "unknown programs" that create "an atmosphere of untrust" and "exacerbate[] the need to secure information and physical resources from any alien code." *Dan*, Abstract; 20 (describing the "Installation of Untrusted Applications"). When the computer receives a new program, its resource capability list (RCL) and certificate, a verifier module "verif[ies] that the code and RCL are valid (not tampered with)." *Id*., 5. Once the code is verified, the RCL manager determines "which subset of parameters within the RCL are to be allowed" and "stores the code and its modified RCL in a secure location." *Id.*, 5-6. "The RCL enforcer is invoked when the code is executed," and it "monitor[s] accesses by the code to ensure that the permissions and resource consumption limits are not violated." *Id.*, 6. Dan also teaches allowing "downloading (or installing from

CDROM) of any code without any certificate as in the conventional systems" and that in these cases, "[s]uch untrusted codes when executed via sandbox environment will be given minimum default resource capabilities." *Id.*, 7.

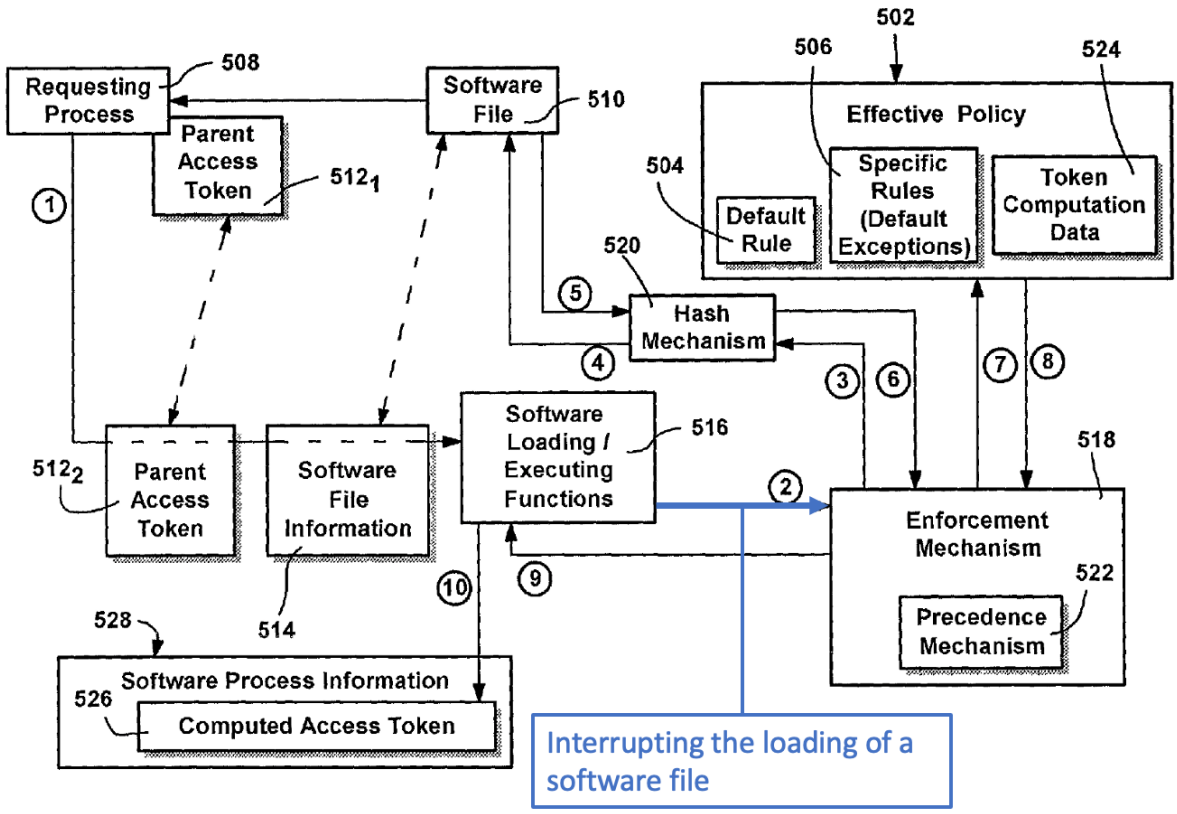
A PHOSITA would have understood that not all applications, including those without certificates, are unknown and/or untrusted. *Declaration*, ¶105. For example, Lambert teaches that "[c]omputer users, and particularly business users, typically rely on a set of programs that are **known and trusted**." *Lambert*, [0003]. "However, in most enterprise environments, particularly in larger environments in which some flexibility is important with respect to what various employees need to do with computers to perform their jobs, users are able to run **unknown and/or untrusted code**, that is, programs outside of the supported set." *Id.* To address the presence of trusted and untrusted programs, Lambert teaches applying different "security levels for execution" including "normal (unrestricted), constrained (restricted) or disallowed (revoked)." *Id.*, [0013]. Per Lambert, "known good code…is allowed to execute" in either "unrestricted (normal) or restricted" modes while "known bad code…will not be allowed to execute at all." *Id.*, [0056].

Lambert proposes determining "access to resources based on an identification (classification) of software that may contain executable content." *Id.*, [0052]. Software "may be identified and classified, with rules set and maintained for the software classifications that automatically and transparently determine the ability of

software to execute." *Id.* "[W]hen software is to be **loaded (e.g., its file opened and some or all thereof read into memory)**, one or more rules are located that correspond to the classification for that software, (or if none corresponds, the default rule), and a selected rule[] is used **to determine whether the software can be loaded**, and if so, to what extent any processes of the software will be restricted." *Id.*; [0014] (describing identifying rules "prior to loading a file").

Figure 5A depicts interrupting the loading of the software file for identification/classification. "When the process 508 **wants to load** and possibly execute the software file 510, it identifies itself and provides information 514 identifying the software file to an appropriate function 516 (of a set) that can open, **load** and/or execute the file 510." *Id.*, [0066]. "**[I]nstead of simply loading the file** and executing any executable code…each of the functions 516 is arranged to first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG. 5A)." *Id.*, [0067]. After the loading of the software file is interrupted by function 516 sending the software file information to the enforcement mechanism 518 in step 2, "the enforcement mechanism consults the effective policy 502 to determine which rule applies for the file 510, and from the rule determines whether to open/execute the file, and if so, the extent of any restricted execution context for the file 510." *Id.* Thus, Lambert teaches *interrupting the loading of*

software so that it can be identified/classified and an appropriate security policy may be applied:



Declaration, ¶108.

Based on the teachings of Lambert, it would have been obvious, and a PHOSITA would have been motivated to configure Dan's RCL enforcer to *interrupt the loading of a new program for operation with the computing device* so that the program can be identified/classified to determine the appropriate security level prior to execution. *Id.*, ¶111. A PHOSITA would have understood that this modification would have had multiple benefits. *Id.*, ¶112. First, this modification would have

enabled Dan's RCL enforcer to block known malware from executing. *Id.*, ¶113. Dan allows "downloading (or installing from CDROM) of any code without any certificate as in the conventional systems." *Dan*, 7. Thus, a user may download known malware that does not have a certificate/RCL to the computing device. *Declaration*, ¶113. Configuring Dan's RCL enforcer to interrupt the loading of the new program would have enabled the RCL enforcer to identify the program as "bad" code prior to it executing and potentially harming the computing device. *Id.*, ¶¶113-114.

Second, this modification would provide solutions for validating conventional programs that lack Dan's certifications/RCLs. *Id.*, ¶115. A PHOSITA would have understood that a program may be obtained directly from a trusted developer instead of a third-party certification agency. *Id.* Incorporating Lambert's identification/classification process would have allowed Dan's sandbox to select an appropriate security policy rather than unnecessarily restricting these trusted programs to one-size-fits-all "minimum default resource capabilities." *Id.*

Another benefit would have been increasing the performance of Dan's sandbox by configuring it not to monitor trusted applications. *Id.*, ¶117. A PHOSITA would have understood that a trusted program that is allowed to execute in an unrestricted manner, as taught by Lambert, would not need to be monitored by the RCL enforcer. *Id.*, ¶116. Dan's RCL enforcer is "invoked for each system call made

by an application." *Dan*, 12. A PHOSITA would have understood that system call interception was costly and impacted the performance of kernel monitoring systems. *Declaration*, ¶117. Configuring the RCL enforcer to not intercept the system calls of a trusted application would have predictably increased the sandbox's performance and would have reduced the computational costs associated with running the sandbox on the computing device. *Id.* A PHOSITA would have been motivated to avoid consuming resources by monitoring trusted programs unnecessarily. *Id.*

For these reasons, a PHOSITA would have been motivated to improve similar computer security methods in the same way by configuring Dan's RCL enforcer to interrupt the loading of a new program on the computing device so that it can be identified/classified prior to execution, as taught by Lambert. *Id.*, ¶118. A PHOSITA would have understood that this modification would not have changed the principle of operation of Dan's client sandbox because Dan's RCL enforcer would still monitor untrusted applications per Dan's stated purpose. *Id.*, ¶119. Rather, this modification would have served to provide additional safeguards and performance improvements to Dan's client sandbox. *Id.*, ¶118. Thus, there would have been a reasonable expectation of success configuring Dan's RCL enforcer to interrupt the loading of a new program for operation with the computing device. *Id.,* ¶119.

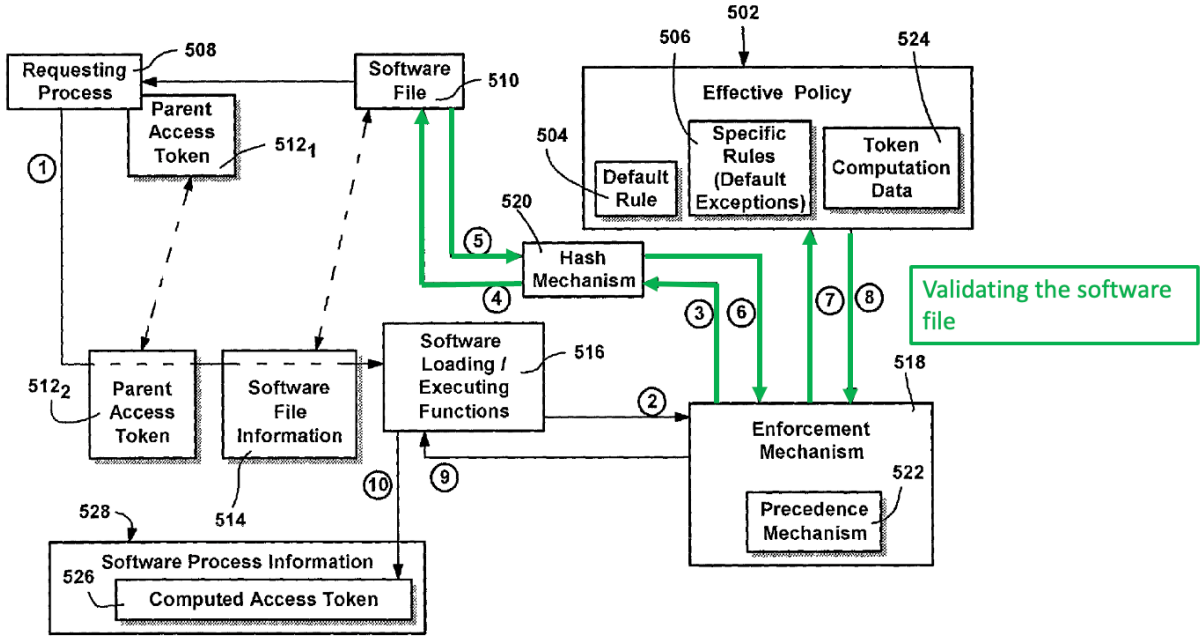
[6(b)] validating the new program;

Lambert teaches that "[t]o identify and classify software,…various selectable classification criteria are made available to an administrator, including **a unique hash value of the file's content (e.g., via an MD5 image hash**)…" *Lambert*, [0053]. Per Lambert, "one way that a software file can be identified (and thus matched to a corresponding rule) is by **a hash of its contents**, which is essentially a fingerprint that uniquely identifies a file regardless of whether the file is moved or renamed." *Id.*, [0055]. "[B]efore opening a file for execution, **a hash value is determined from that file's contents**, and the set of hash value information (e.g., in the policy object) searched to determine if a rule exists for that hash value. If so, that rule provides the security level for the software." *Id.*

Regarding the *validating* step, the '137 Patent discloses "the validation module can represent the **MD5 software module** commonly known to those in the art," which "calculates a checksum for each allowed program and that checksum is stored for later comparison."*'137 Patent*, 8:13-17. Lambert similarly teaches comparing the MD5 hash of a software file to a set of stored hash values for known software files having rules associated with them. *Lambert*, [0053], [0055]. Per Lambert, "an administrator can set a rule for any file based on its unique hash value, with the rule specifying whether (and if so, how) a software file having that hash value will execute." *Id.*, [0056]. "In the case of known bad code, the administrator

may set the rule to specify that any file with that hash will not be allowed to execute at all (i.e., 'disallowed' or 'revoked' status)." *Id.* "Since the rule along with the hash value is maintained or distributed to the machine, (e.g., via a policy object), whenever a request to open that file is received, a hash of the file matches the saved hash and rule, and that file will not be run." *Id.* "Alternatively, in the case of known good code that is allowed to execute, a hash rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent)." *Id.* If no rules apply, configurable default rules apply. *Id.*, [0052],[0058],[0065],[0068],[0083]. Thus, Lambert uses the software file's hash to validate whether it should be allowed to execute in an unrestricted manner, restricted manner, or not at all. *Declaration*, ¶122.

Lambert's *validating* step is also depicted in Figure 5A. Once the enforcement mechanism receives the software file's information from the software loading function 516, it calls "a hash function 520 or the like to obtain a hash value from the contents of the software file 510, as generally represented in FIG. 5A by the arrows labeled with circled numerals three (3) to six (6)." *Lambert,* [0068]. "[A] suitable hash function 520 provides a unique fingerprint of the software file 510, while the effective policy 502 may have a specific rule for that hash." *Id.* Arrows seven (7) and eight (8) "represent[] accessing the policy to get the rule or rules" by looking for a rule corresponding to "the hash value." *Id.* Therefore, steps 3-8 of Figure 5A collectively qualify as *validating* the software within the meaning of the '137 Patent:



Declaration, ¶123.

For the reasons discussed above, it would have been obvious to a PHOSITA to configure Dan's sandbox to identify and classify applications as bad (i.e., no execution), good (unrestricted execution), or untrusted (restricted execution) as taught by Lambert. *See* Claim 6(a). Since Lambert's identification/classification process qualifies as *validating*, Dan's RCL enforcer modified in view of Lambert also renders obvious *validating the new program* for the same reasons. *Id.*

It would have also been obvious, and a PHOSITA would have been motivated to configure Dan's RCL enforcer to validate the new application by comparing its hash value to a list of hashes for known applications, as taught by Lambert. *Declaration*, ¶125. A PHOSITA would have understood that comparing a hash of

the new application to a list of hashes for known applications would have been a reliable way to validate whether the new application is trusted or untrusted. *Id*. Therefore, a PHOSITA would have been motivated to combine the prior art elements of Dan's RLC enforcer and Lambert's method of using hashes to identify and classify a software file according to known methods to yield the predictable result of allowing Dan's sandbox to validate the new application by identifying it as bad (i.e., no execution), good (unrestricted execution), or untrusted (restricted execution) even when no certificate is provided. *Id.*

The '137 Patent acknowledges that the MD5 software modules were "commonly known to those in the art." *'137 Patent*, 8:14-15. A PHOSITA would have understood that an MD5 hash is a well-known type of cryptographic hash. *Declaration*, ¶126. Therefore, there would have been a reasonable expectation of success configuring Dan's RCL enforcer to validate the new application using a "commonly known" MD5 hash. *Id.*

[6(c)] if the new program is validated, permitting the new program to continue loading and to execute in connection with the computing device;

Petitioner construes this limitation to mean "if the new program is validated, then it is not monitored while it loads and executes in connection with the computer device." *See* Section II.C.1.

Lambert describes a case where the software is *validated* in that it is identified as "good code" that is allowed "unrestricted" execution. *Lambert*, [0056] ("[I]n the

case of known good code that is allowed to execute, a hash rule can specify how it will be executed, e.g., unrestricted (normal)…"); *see also, id.,* [0008].

Unrestricted execution is different from restricted execution in that a restricted execution environment "reduces software's access to resources, and/or removes privileges, relative to a user's normal access token." *Id.*, Abstract. When it is determined that software should be restricted, "the enforcement mechanism 518 can compute and associate a restricted token with the software file 510, whereby if creation of a process is requested for the software file, (it includes executable content), any process of the software file is restricted in its access rights and privileges according to the restricted token." *Id.*, [0070]. When the user requests to load a software file, "the enforcement mechanism 518 completes the call and returns information back to the software function 516 that called it, the returned information essentially including an instruction on whether to open the file, and if so, the token to associate with the file." *Id.*, [0071]. "If a token is returned, it may be the parent token (unchanged) if no restricted execution environment is required by the rule, or a restricted token that establishes a restricted execution environment/context for the processes of the software file 510." *Id.* "If the computed token 526 is a restricted token, the process of the software file can now execute, but only in association with the restricted token, thus providing the rule-determined secure execution environment." *Id.* Lambert's Figure 3 and the related disclosure describe how a

restricted program is monitored based on its restricted token. *Declaration*, ¶129, ¶88 (*citing Lambert*, [0042]-[0043], Fig. 3). Specifically, every time a restricted program attempts to access an object (e.g., a file object), access is approved or denied based on the restrictions in the program's restricted token. *Id.* Thus, Lambert teaches that restricted mode requires monitoring the restricted program, which is not required in unrestricted mode. *Id*., ¶130.

For these reasons, Lambert teaches determining that the software is *validated* for unrestricted execution, which results in it not being monitored while it loads and executes on the computer. *Id.*

As discussed above, it would have been obvious to a PHOSITA to configure Dan's RLC enforcer to validate the new program by comparing its hash to the hashes of other known programs. *See* Claim 6(b). For the same reasons, it would have been obvious to configure Dan's RLC enforcer to determine *if the program is validated* if its hash matches the hash of a known trusted application. *Id.* As discussed above, configuring the RCL enforcer to not monitor a trusted application would have predictably improved the performance of Dan's sandbox. *See* Claim 6(a). A PHOSITA would have understood that a trusted application would not pose a serious security risk to the computing device and monitoring a trusted application would be a waste of the computing device's resources. *Declaration*, ¶132. Therefore, a PHOSITA would have been motivated to improve similar computer-implemented

security methods in the same way by configuring Dan's RLC enforcer to allow a trusted application to load and execute without monitoring the application's use of the client system's resources, thereby improving the sandbox's performance.[4] *Id.*

Modifying Dan's RLC enforcer to not monitor trusted applications would also not have changed the principle of operation because Dan would still monitor untrusted applications per Dan's stated purpose. *Id.*, ¶133. Given these factors and the predicably of computer programming at the time of the '137 Patent, a PHOSITA would have had a reasonable expectation of success configuring Dan's RLC enforcer to not monitor a trusted application while it is loading and executing on the computing device. *Id.*

[6(d)(i)] if the new program is not validated, monitoring the new program while it loads and executes in connection with the computing device,

Lambert also teaches a case where the software is *not validated* and must be executed "in a restricted execution environment." *Lambert*, [0008]. For any software that does not have a specific rule associated with it, "an administrator may set up a default rule that generally restricts files[.]" *Id.*, [0058]. Lambert teaches that

[4] Petitioner notes that in co-pending district court litigation PO has alleged this limitation means "that validated programs can still be monitored, but it is not required." *Taasera's Claim Construction Brief,* 1. Thus, the prior art also teaches this limitation under PO's broader interpretation of the claim.

applications executing in a restricted execution environment are monitored. *See* Claim 6(c).

Dan's sandbox monitors the new program while it loads and executes on the computer. Specifically, the RCL enforcer "ensures that the permissions and resource consumption limits specified in the RCL are not violated." *Dan*, 9. "The module for enforcing system resource capabilities monitors two types of resources: physical and logical." *Id.* "Access control information about physical and logical resources of the processing executing an alien application code is maintained in data structures called the Runtime Physical Resources Table (RPRT) and the Runtime Logical Resources Table (RLRT)." *Id.* "**When an application is initially loaded for execution**, the RPRT and RLRT data structures are initiated from the information specified in the RCL configuration file associated with this application." *Id.* "**During process execution**, the RCL enforcer references and updates the information stored in the RPRT data structure using the algorithm outlined in Figure 5 to allow or deny the request for accessing physical resources." *Id.* "Similar to the RPRT, access control information for the logical resources is maintained in a data structure called the Runtime Logical Resources Table (RLRT)." *Id*., 12. "The logical resources constitute file, directories, communication ports and various other system service calls." *Id.* "The RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system." *Id.* "This monitoring requires that the RCL enforcer

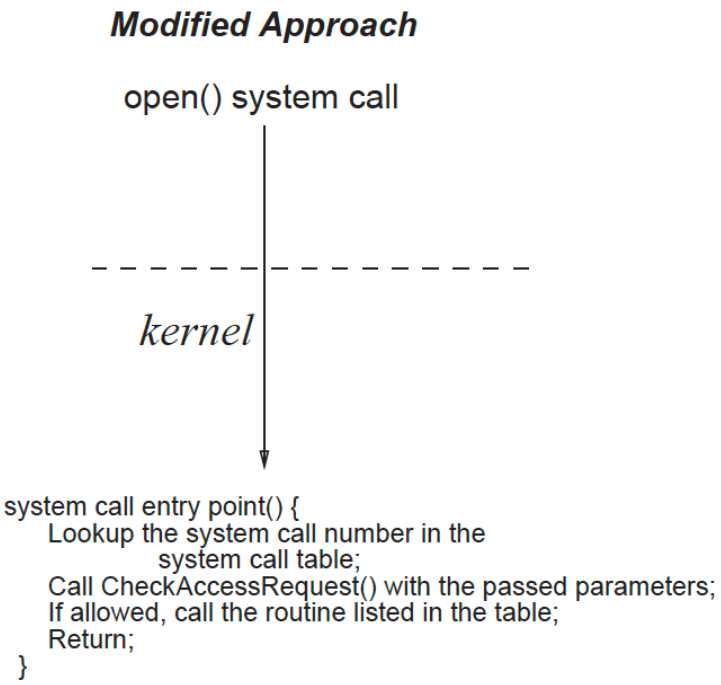
be invoked for each system call made by an application." *Id.* "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT data structures and either allows or disallows the call from continuing." *Id.* Thus, Dan teaches *monitoring the new program while it loads and executes in connection with the computing device*. *Declaration*, ¶¶135-136.

When modifying Dan in view of Lambert, it would have been obvious, and a PHOSITA would have been motivated to configure Dan's RCL enforcer to perform monitoring of the new program while it loads and executes on the host computer when the program fails validation for unrestricted execution, including for programs that lack Dan's certificates. *Id.*, ¶137. A PHOSITA would have understood this modification simply amounts to making the monitoring conditional on the validation outcome and to not perform monitoring on new programs that are determined to be trusted, as discussed with regard to Claim 6(c). *Id.* Dan teaches monitoring untrusted new programs while they load and execute on the client device. *Id.* It would have been obvious to configure the sandbox to perform monitoring on new applications, as described by Dan, based on the outcome of the validation. *Id.* A PHOSITA would have been motivated to combine prior art elements according to known methods to yield the predictable result of Dan's sandbox monitoring the untrusted application when it fails the validity check. *Id.* Since the purpose of Dan's sandbox is to monitor untrusted applications, there would have been a reasonable expectation of success

configuring Dan's RCL enforcer to monitor the new application when it fails the validity check. *Id.*

[6(d)(ii)] wherein the step of monitoring the new program while it executes is performed at the operating system kernel of the computing device.

Dan teaches that "[t]he RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system." *Dan*, 12. Dan teaches three different approaches to monitoring system calls and selects "system call modification." *Id*, 14. "In this approach, the **operating system's call interface** is modified to call the CheckAccessRequest() upon entry to verify the parameters of the system call." *Id.* "Thus, regardless of how the applications are structured, **the system call entry point in the operation systems** will ensure consistent security checks for all applications that are executed on the machine." *Id.* As shown in Figure 10, this the system call modification technique performs monitoring at the operating system kernel of the client system:



Id., 15; *Declaration*, ¶138.

Thus, Dan teaches that *the step of monitoring the new program while it executes is performed at the operating system kernel of the computing device*. *Declaration*, ¶139.

The CAFC requires an obviousness analysis when "selecting from large lists of elements in a single reference." *In re Stepan*, 868 F.3d 1342, 1346 n.1 (Fed. Cir. 2017). While a list of three alternative implementations is not a "large list of elements," it nonetheless would have been obvious, and a PHOSITA would have been motivated, to implement Dan's sandbox using the expressly suggested system call modification method. *Declaration*, ¶140. Given Dan's express teaching, suggestion, and motivation, a PHOSITA would have been motivated to implement

Dan's sandbox using the system call modification method. *Id.* Since Dan teaches that this is the most consistent way to enforce security, there would have been a reasonable expectation of success configuring Dan's sandbox to use the system call modification method. *Id*., ¶141.

2. Claim 8

8. The method of claim 6, wherein the step of monitoring the new program comprises intercepting a signal from the computing device's operating system.

Dan teaches monitoring the new application by *intercepting a signal from the computing device's operating system* by modifying the operating system's call interface. *See* Claim 6(d)(ii).

3. Claim 9

9. The method of claim 6, wherein the step of validating the new program comprises determining whether the new program corresponds with an approved program.

It would have been obvious to a PHOSITA to configure Dan to incorporate Lambert's *validating* step. *See* Claim 6(b). As part of this *validating* step, Lambert teaches comparing the hash of a software file to the hashes of other known software files to determine if the software file being loaded corresponds to an approved program that is allowed to execute in unrestricted mode. *Id.* Thus the combination of Dan and Lambert also teaches *wherein the step of validating the new program comprises determining whether the new program corresponds with an approved program*. *Id.*

4. Claim 12

12. The method of claim 6, wherein the step of monitoring the new program comprises controlling the files the new program attempts to access during execution of the new program.

Dan teaches that "[t]he RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system" where "logical resources constitute **file**, directories, communication ports and various other system service calls." *Dan,* 12, 8 (restrictions on files that can be accessed in read-write mode), 9 ("files to which access needs to be restricted"), 10 ("Files/directories"), 14 (system calls to file system). After the RCL enforcer is invoked by a system call attempting to access a logical resource, such as a file, "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT data structures and either allows or disallows the call from continuing." *Id.*, 12. Thus, Dan teaches *wherein the step of monitoring the new program comprises controlling the files the new program attempts to access during execution of the new program*. *Declaration*, ¶¶144-145.

5. Claim 25

***25[Pre]. A computer-implemented method for performing security for a computer device during a pre-execution phrase* [sic] *comprising the steps of:*[5]**

[5] There appears to be a typographical error in the preamble of Claim 25. If the Board determines that the preamble of Claim 25 is limiting, Petitioner interprets "phrase" to mean "phase."

As discussed above regarding Claim 6, Dan teaches *a computer-implemented method for performing security for a computer device*. *See* Claim 6[Pre]. As set forth below, the combination of Dan and Lambert embodies a *computer-implemented method for performing security for a computer device during a pre-execution phase*.

[25(a)] identifying an allowed program that is permitted to execute with the computing device;

Dan primarily focuses on unknown/untrusted programs and automatically assumes that any program without a certificate/RCL should be given with minimum resource capabilities. *See* Claim 6(a). However, a PHOSITA would have understood that not all applications without certificates are unknown and/or untrusted. *Id*.

Lambert teaches a pre-execution step where the system administrator identifies a software file as being allowed to execute on a computing device in either a restricted or unrestricted mode. "Via a user interface or the like operably connected to a suitable hash mechanism, an administrator can select any software file, obtain a hash value from its contents, and enter and associate a rule (that specifies a security level) with that hash value." *Lambert*, [0055]. "Multiple files can have their corresponding hash values and rules maintained in a policy object, or otherwise made available to the system, whereby for any given instance of any file, the set of maintained hash values can be queried to determine whether a rule exists for it." *Id.* "In this manner, an administrator can set a rule for any file based on its unique hash value, with the rule specifying whether (and if so, how) a software file having that

hash value will execute." *Id.*, [0056]. "[I]n in the case of known good code that is **allowed to execute**, a hash rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent)." *Id.* Thus, Lambert teaches a user interface that allows the administrator to *identify an allowed program that is permitted to execute with the computing device*. *Declaration*, ¶148.

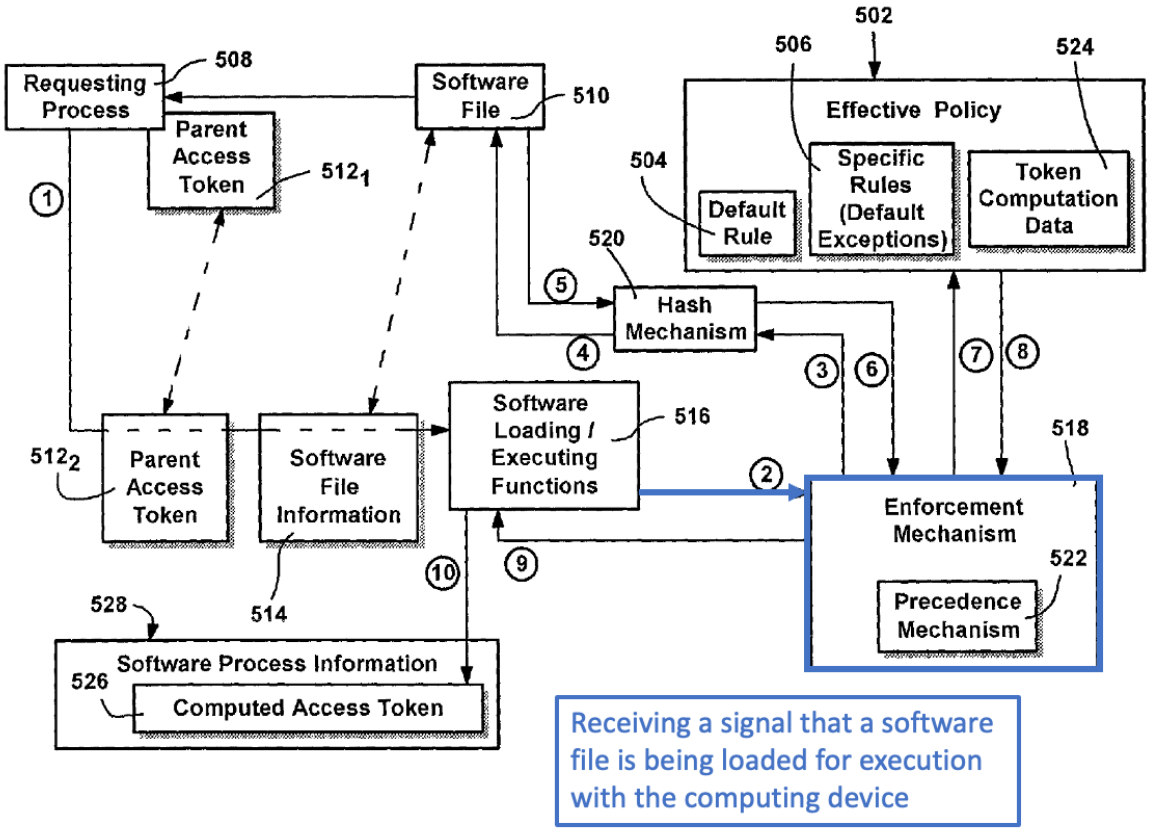
Based on the teachings of Lambert, it would have been obvious, and a PHOSITA would have been motivated to configure Dan's system to include a user interface that allows the system administrator to *identify an allowed program that is permitted to execute with the computing device*. *Id.*, ¶149. Specifically, it would have been obvious to a PHOSITA to configure Dan's system to allow the system administrator to identify "good" programs that are allowed to execute in an unrestricted or less restricted manner even if they do not have certificates/RCLs. *Id*. Configuring Dan's system to include a user interface where an administrator can identify allowed applications would have enabled the administrator to define security policies for applications that do not have certificates/RCLs. *Id*. It was a well-known practice for administrators to maintain lists or databases of hashes associated with known applications. *Id.* A PHOSITA would have understood that allowing an administrator of Dan's system to identify allowed programs in advance would have predictably made it easier to ascertain whether a given application should be allowed to execute and, if so, what (if any) restrictions should apply in the absence of a

certificate/RCL. *Id*. Therefore, a PHOSITA would have been motivated to combine the prior art elements of Dan's security method with Lambert's step of allowing a system administrator to identify an allowed program that is permitted to execute to yield the predictable result of a security system that applies appropriate security policies to applications that do not have certificates/RCLs. *Id.*

Dan teaches providing utilities for the administrator, including an InstallCop utility that the administrator uses to edit configuration files and manage the system-wide installation of software applications. *Dan*, 20 ("[T]he InstallCop can allow the system administrator to edit the system default configuration file before the application is made available to users."). Thus, a PHOSITA would have understood that adding another utility for the system administrator to identify programs that are allowed to execute on the computing device would have only required minor software changes. *Declaration*, ¶150. Allowing an administrator to identify applications that are allowed to execute in either unrestricted or restricted mode would not have changed the principle of operation of Dan's sandbox since it would still execute untrusted code having associated restrictions. *Id.* Given these factors, there would have been a reasonable expectation of success configuring Dan's sandbox to include a user interface that allows a system administrator to *identify an allowed program that is permitted to execute with the computing device*. *Id.*

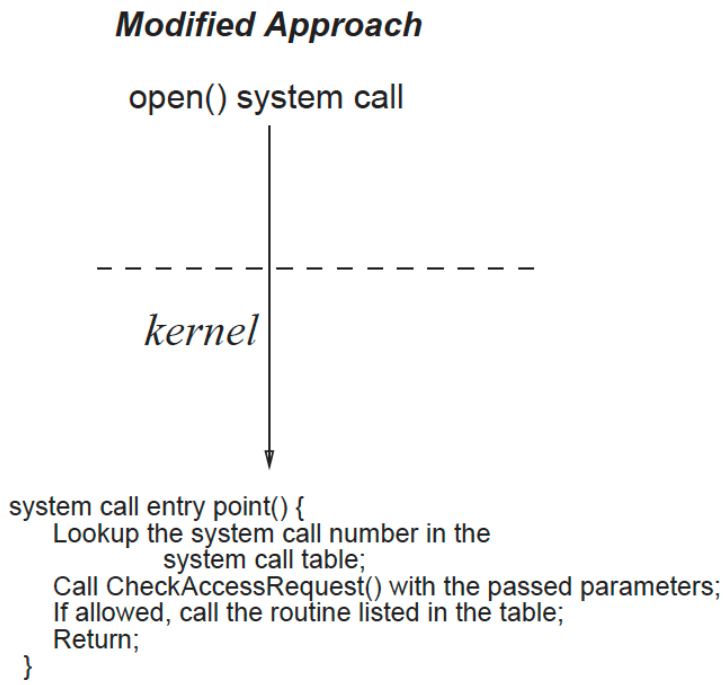
[25(b)] receiving a signal that a new program is being loaded for execution with the computing device;

Dan teaches *a new program*. *See* Claim 6(a). Lambert teaches *receiving a signal that* a software file *is being loaded for execution with the computing device*. "[I]n FIG. 5A, a process 508 such as a process of a user or application requesting to open a software file 510 has an access token 512_1 associated with it[.]" *Lambert*, [0066]. "**When the process 508 wants to load and possibly execute the software file 510**, it identifies itself and provides information 514 identifying the software file **to an appropriate function 516** (of a set) that can open, **load and/or execute** the file 510." *Id.* "For example, this can be accomplished by placing an application programming interface (API) call to the operating system, wherein the file information 514, typically in the form a path and filename is passed, as a parameter." *Id.* "Further note that **such function calls are already the typical way in which files are loaded**, and thus the process 508…need not be modified in any way." *Id.* "[I]nstead of simply loading the file and executing any executable code therein with the parent token's rights and privileges, each of the functions 516 is arranged to **first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG. 5A)**." *Id.*, [0067]. Thus, Lambert teaches that the enforcement mechanism 518 *receives a signal that a* software file *is being loaded for execution with the computer device* from the software opening function 516:



Declaration, ¶152.

The combination of Dan and Lambert renders obvious the step of *receiving a signal that a new program is being loaded for execution with the computing device*. *Id*. ¶153. Dan teaches an RLC enforcer module that "monitors all system calls" and further teaches that "the **operating system's call interface** is modified to call the CheckAccessRequest() upon entry to verify the parameters of the system call." *Id.*, 12, 14. In one example, Dan teaches modifying an "open() system call" to send a signal to the RCL enforcer by calling the RCL enforcer's CheckAccessRequest() function instead of opening the file:



Dan, 14-15; *Declaration*, ¶153.

This is similar to Lambert's teaching of the operating system function 516 passing information about the software file to the enforcement mechanism 518 rather than loading the software file. *Declaration*, ¶154.

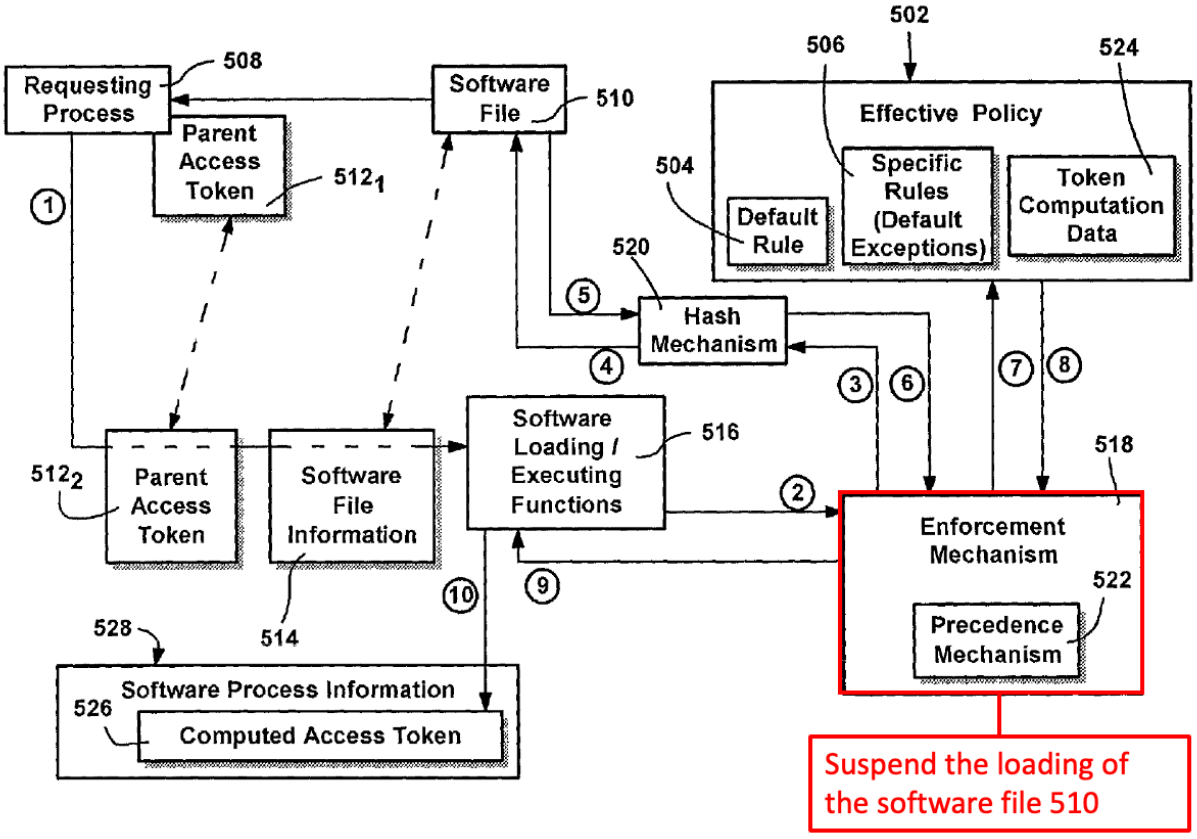
Based on the teachings of Lambert, it would have been obvious, and a PHOSITA would have been motivated to configure Dan's RCL enforcer to *receive a signal that a new application is being loaded for execution with the computing device* via the system call functions associated with loading and executing a new program. *Id.,* ¶155. For the reasons described above, it would have been obvious to a PHOSITA to configure Dan's system to allow an administrator to identify applications that are allowed to execute and define their corresponding security

policies to account for applications that do not have certificates/RCLs. *See* Claim 25(a). For the same reasons, it would have also been obvious to configure the RCL enforcer to receive a signal when the new application is being loaded so that the RCL enforcer may determine what security policy should apply. *Id.*; *Declaration*, ¶155. A PHOSITA would have understood that configuring Dan's RLC enforcer to receive a signal would have predictably allowed it to determine whether an application without a certificate is allowed to execute and, if so, whether it may execute in a restricted or unrestricted manner, prior to loading. *Id*. Therefore, a PHOSITA would have been motivated to combine prior art elements according to known methods to yield the predictable result of allowing Dan's sandbox to determine whether the new application needs to be monitored prior to it being loaded and executed on the computing device even if it does not have a certificate/RCL. *Id.*

Since Dan's RCL enforcer module is already configured to monitor "all system calls" [*Dan*, 12], there would have been a reasonable expectation of success configuring Dan's RCL enforcer to *receive a signal that a new application is being loaded for execution with the computing device* from the operating system call functions associating with loading and executing a new program, as taught by Lambert. *Id.*, ¶156.

[25(c)] suspending the loading of the new program;

Per Lambert, "when software is to be **loaded (e.g., its file opened and some or all thereof read into memory)**, one or more rules are located that correspond to the classification for that software, (or if none corresponds, the default rule), and a selected rules is used **to determine whether the software can be loaded**, and if so, to what extent any processes of the software will be restricted." *Lambert*, [0052]; *see also, id.*, [0014] (describing finding an associated rule prior to loading the software file). Once the enforcement mechanism 518 receives the software file information 514 from the software loading/execution functions 516, "the enforcement mechanism consults the effective policy 502 to determine which rule applies for the file 510, and from the rule determines whether to open/execute the file, and if so, the extent of any restricted execution context for the file 510." *Id.*, [0067]. Thus, Lambert teaches *suspending the loading* of the software so that the enforcement mechanism 518 can determine the rules that apply to the software:



Declaration, ¶¶158-159.

The combination of Dan and Lambert teaches *suspending the loading of the new program*. *Id.*, ¶160. For the same reasons as discussed above, configuring Dan's sandbox to identify and classify a new program would have allowed it to select the appropriate security policy even if it does not have a certificate/RCL. *See* Claim 25(b). A PHOSITA would have been motivated to improve similar computer-implemented security methods in the same way by configuring Dan's RLC enforcer to suspend the loading of the new application until the RLC enforcer identifies the application and the security rules that apply to it. *Declaration*, ¶160.

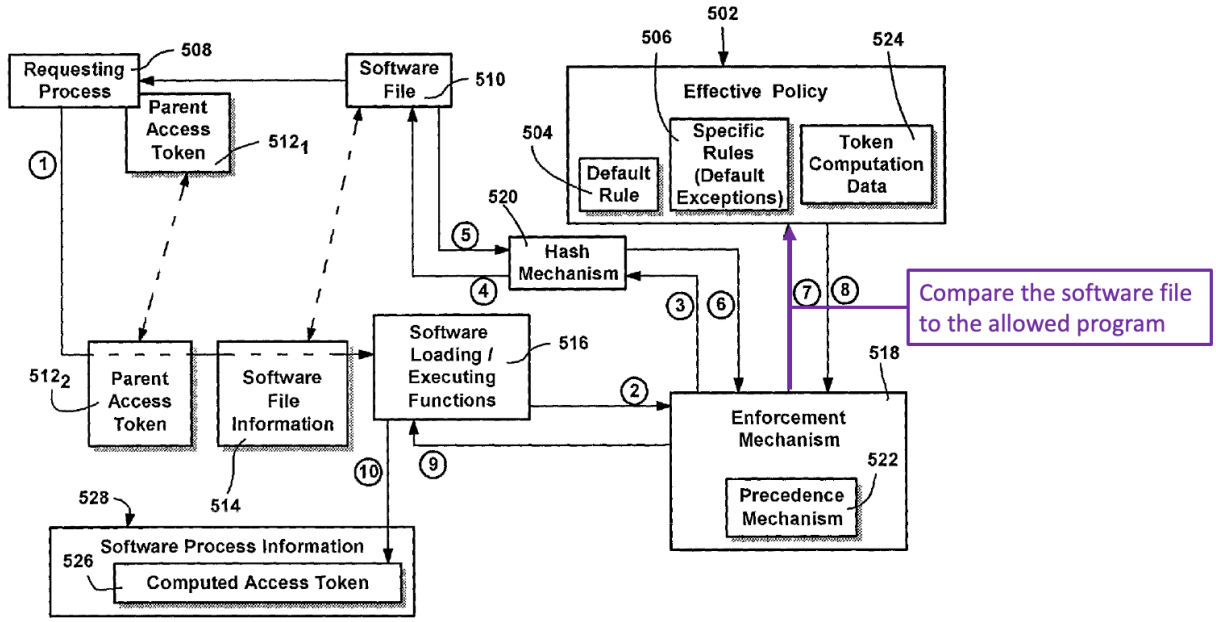
As described above, Dan's RCL enforcer module is configured to monitor "all system calls." *Dan*, 12. Thus, there would have been a reasonable expectation of success configuring Dan's RCL enforcer to suspend the loading of the new program upon receipt of a signal from the operating system call interface to allow the RCL enforcer to determine the appropriate security policy for the application even if it does not have a certificate/RCL. *Declaration*, ¶161.

[25(d)] comparing the new program to the allowed program; and

Lambert teaches that the enforcement mechanism 518 "consults the effective policy 502 to determine which rule applies for the file 510, and from the rule determines whether to open/execute the file, and if so, the extent of any restricted execution context for the file 510." *Lambert*, [0067]. "[T]he enforcement mechanism 518 may call (or include in its own code) a hash function 520 or the like to obtain a hash value from the contents of the software file 510, as generally represented in FIG. 5A by the arrows labeled with circled numerals three (3) to six (6)." *Id.*, [0068]; *see also, id.,* Fig. 5A. "[A] suitable hash function 520 provides a unique fingerprint of the software file 510, while the effective policy 502 may have a specific rule for that hash." *Id.* For example, Lambert teaches that the hash is "a unique hash value of the file's content (e.g., via an MD5 image hash)." *Id.*, [0053].

To determine if the software file's hash matches the hash of a known software file, "the set of maintained hash values can be queried to determine whether a rule

exists for it." *Id.,* [0055]. "[B]efore opening a file for execution, a hash value is determined from that file's contents, and the set of hash value information (e.g., in the policy object) [is] searched to determine if a rule exists for that hash value." *Id.* The set of maintained hash values includes the hash value of the "known good code that is allowed to execute" (i.e., *the allowed program*) that was previously identified by the system administrator. *See* Claim 25(a). "Note that while FIG. 5A essentially represents accessing the policy to get the rule or rules via arrows labeled seven (7) and eight (8),…e.g., to look first for a rule for the hash value[.]" *Lambert*, [0068]. Thus, Lambert teaches *comparing* the hash value of the software file to a set of maintained hash values, including the hash value of *the allowed program*.

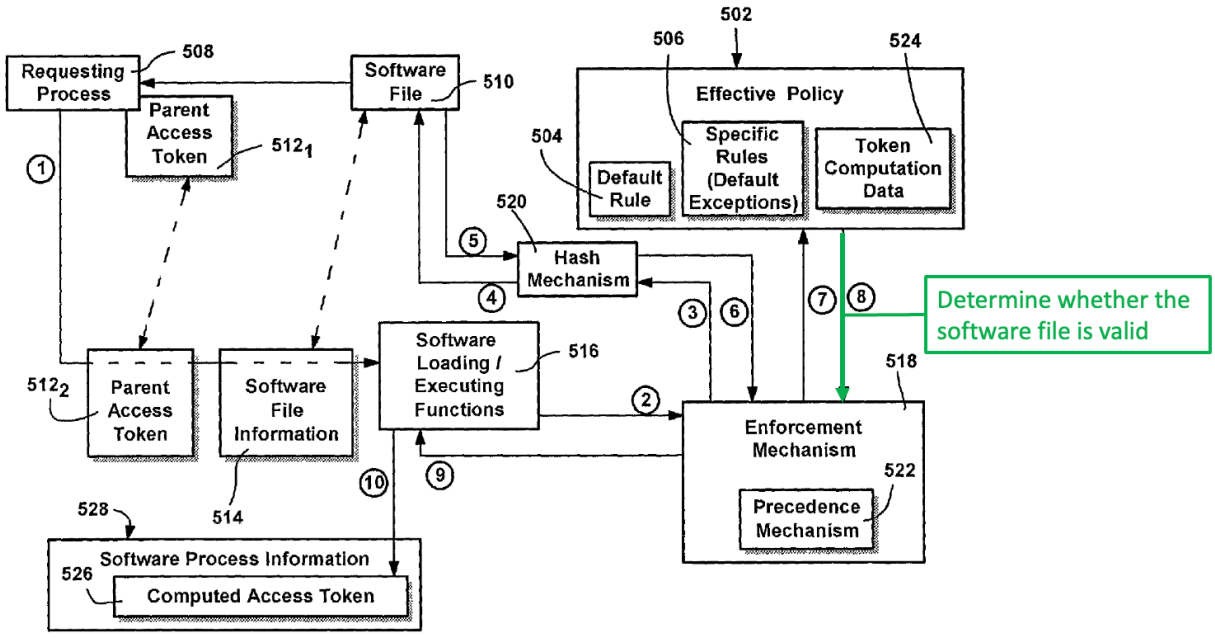


Declaration, ¶163.

The combination of Dan and Lambert teaches *comparing the new program to the allowed program*. *Id.*, ¶164. For the reasons discussed above, it would have been obvious to a PHOSITA to configure Dan's sandbox to be able to identify/classify a new program to allow the sandbox to select the appropriate security policy even if the program does not have a certificate/RCL. *See* Claim 25(b). Based on the teachings of Lambert, it would have been obvious, and a PHOSITA would have been motivated to configure Dan's RCL enforcer to determine if the new application is an allowed application by comparing its hash value to a list of hashes for known applications (including the allowed program). *Declaration*, ¶164. A PHOSITA would have understood that using a hash of the new application and comparing it to a list of hashes for known applications would have been a reliable way to identify the application. *Id.* Therefore, a PHOSITA would have been motivated to combine prior art elements according to known methods to yield the predictable result of allowing Dan's RCL enforcer module to determine whether the new application corresponds to an allowed application. *Id.* There would have been a reasonable expectation of success configuring Dan's RCL enforcer module to validate the new application using "commonly known" MD5 hashes. *Id.*, ¶165 (citing *'137 Patent*, 8:14-15).

[25(e)] determining whether the new program is valid;

Lambert teaches determining whether the software file is valid based on comparing the software file's hash value to the set of maintained hash values "to determine whether a rule exists for it." *Lambert*, [0055]. "[B]efore opening a file for execution, a hash value is determined from that file's contents, and the set of hash value information (e.g., in the policy object) searched to determine if a rule exists for that hash value. If so, that rule provides the security level for the software." *Id.* "[I]n the case of known good code that is allowed to execute, a hash rule can specify how it will be executed[.]" *Id.*, [0056]. As shown below in Fig. 5A, the enforcement mechanism 518 determines whether the software file is valid based on the hash rule returned from the effective policy 502:



Declaration, ¶166.

The combination of Dan and Lambert renders obvious *determining whether the new program is valid*. *Id.*, ¶167. Based on the teachings of Lambert, it would have been obvious, and a PHOSITA would have been motivated to configure Dan's RCL enforcer to determine whether the new program is valid based on a rule associated with the new program's hash value. *Id.* Specifically, it would have been obvious to configure Dan's RCL enforcer to determine if the new program is valid (even in the absence of a certificate/RCL) and does not need monitoring or if the new application is not valid and must operate in a restricted environment where its access to the physical and logical resources of the computing device is monitored. *Id.* A PHOSITA would have understood that using a rule associated with the new application's hash value to determine whether the new application is valid would have been an effective and reliable way to distinguish trusted applications that do not need to be monitored from untrusted applications that do need to be monitored even if the program does not have an associated certificate/RCL. *Id.* Therefore, a PHOSITA would have been motivated to combine prior art elements according to known methods to yield the predictable result of allowing Dan's RCL enforcer module to determine whether the new application is valid/trusted and does not need to be monitored or whether the new application is not valid/untrusted and needs to be monitored. *Id.*

A PHOSITA would have understood that configuring Dan's RCL enforcer to determine whether the new application is valid would have represented a minor software change. *Id.,* ¶168. Thus, there would have been a reasonable expectation of success configuring Dan's RCL enforcer to determine whether the new application is valid based on a rule associated with its hash value. *Id.*

[25(f)] if the new program is valid, permitting the new program to execute on the computing device; and

See Claim 6(c).

[25(g)(i)] if the new program is not valid, monitoring the new program while allowing it to execute on the computing device,

See Claim 6(d)(i).

[25(g)(ii)] wherein the step of monitoring the new program while allowing it to execute is performed at the operating system kernel of the computing device.

See Claim 6(d)(ii).

B. Ground 2: The Combination of Lambert, Dan, and Schmid Renders Claims 1 and 2 Obvious

1. Claim 1

1[Pre]. A system for managing security of a computing device comprising:

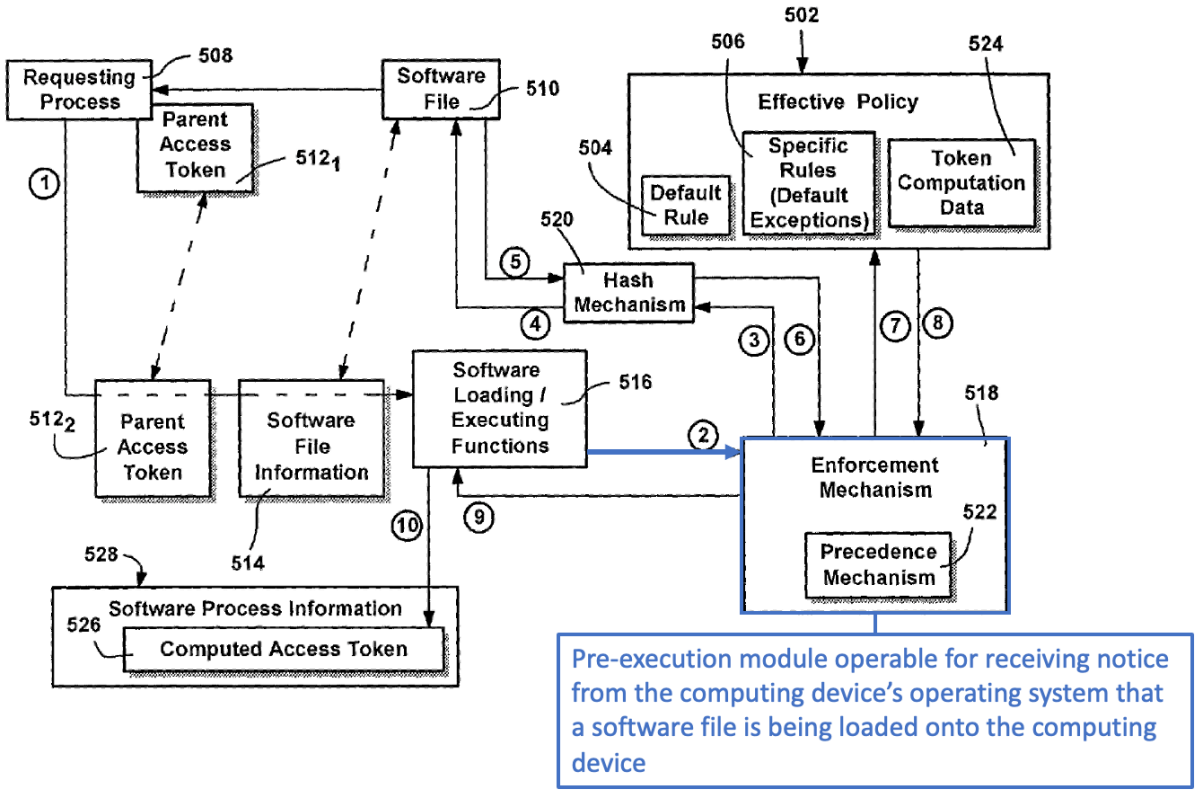
Lambert teaches "[a] system…that automatically, transparently and **securely controls software execution** by identifying and classifying software, and locating a rule and associated **security** level for executing executable software." *Lambert*, Abstract. Lambert's system operates on a "computing system environment 100," which includes "a general purpose computing device in the form of a computer 110."

Id., [0026], [0029], Fig. 1. Thus, Lambert teaches *a system for managing security of a computing device*. *Declaration*, ¶172.

[1(a)] a pre-execution module operable for receiving notice from the computing device's operating system that a new program is being loaded onto the computing device;

Lambert's system includes an enforcement mechanism 518 that *receives notice from the computing device's operating system that* a software file 510 *is being loaded onto the computing device*. "[I]n FIG. 5A, a process 508 such as a process of a user or application requesting to open a software file 510 has an access token 512_1 associated with it[.]" *Lambert*, [0066]. "When the process 508 wants to **load and possibly execute the software file 510**, it identifies itself and provides information 514 identifying the software file to an appropriate function 516 (of a set) that can open, **load and/or execute** the file 510." *Id.* "For example, this can be accomplished by placing an application programming interface (API) call to **the operating system**, wherein the file information 514[.]" *Id.* "Note that in a typical implementation, the **software opening function 516 comprises an operating system component**[.]" *Id.* "**[I]nstead of simply loading the file and executing** any executable code…each of the functions 516 is arranged to **first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG. 5A)**." *Id.*, [0067]. Thus, Lambert's enforcement mechanism 518 is *a pre-execution module operable for receiving notice* (i.e., software file information 514) *from the computing device's*

operating system (i.e., software loading/executing functions 516) *that a* software file *is being loaded onto the computing device*:



Declaration, ¶173.

Lambert teaches that the software file 510 is a *program*. *Declaration*, ¶174. For example, Lambert teaches an example where the software file is an "e-mail program." *Lambert*, [0053]. A PHOSITA would understand that Lambert does not distinguish between old and new programs and encompasses both. *Declaration*, ¶174. For the reasons discussed above, Dan teaches *a new program* that is downloaded to a host computer "via network, diskette, CD-ROM, satellite, etc." *See* Section V.A.1 at Claim 6(a); *Dan*, 20.

Based on the teachings of Dan, it would have been obvious to a PHOSITA that Lambert's software file 510 would correspond to a newly downloaded program. *Declaration*, ¶175. Lambert acknowledges that "with the rise in the usage of networks, email and the Internet for business computing, users find themselves exposed to a wide variety of executable programs including content that is not identifiable as being executable in advance." *Lambert*, [0007]. A PHOSITA would have understood that the computing device would receive software files via networks, email, and the Internet and that, in this instance, the software file would represent *a new program*. *Declaration*, ¶175.

A PHOSITA would have understood that Lambert's enforcement mechanism is operable to receive notice when any software file is being loaded for execution on the computing device. *Id.*, ¶176. For these reasons, a PHOSITA would have been motivated to combine prior art elements according to known methods to yield the predictable result of notifying Lambert's enforcement mechanism when a new program is loading. *Id.* There would have been a reasonable expectation of success since Lambert's enforcement mechanism is notified when any software file (new or otherwise) is loading. *Id.*

[1(b)] a validation module coupled to the pre-execution monitor operable for determining whether the program is valid;

Petitioner construes *the pre-execution monitor* as *the pre-execution module*. *See* Section II.C.2. Lambert teaches an effective policy 502 that is coupled to the

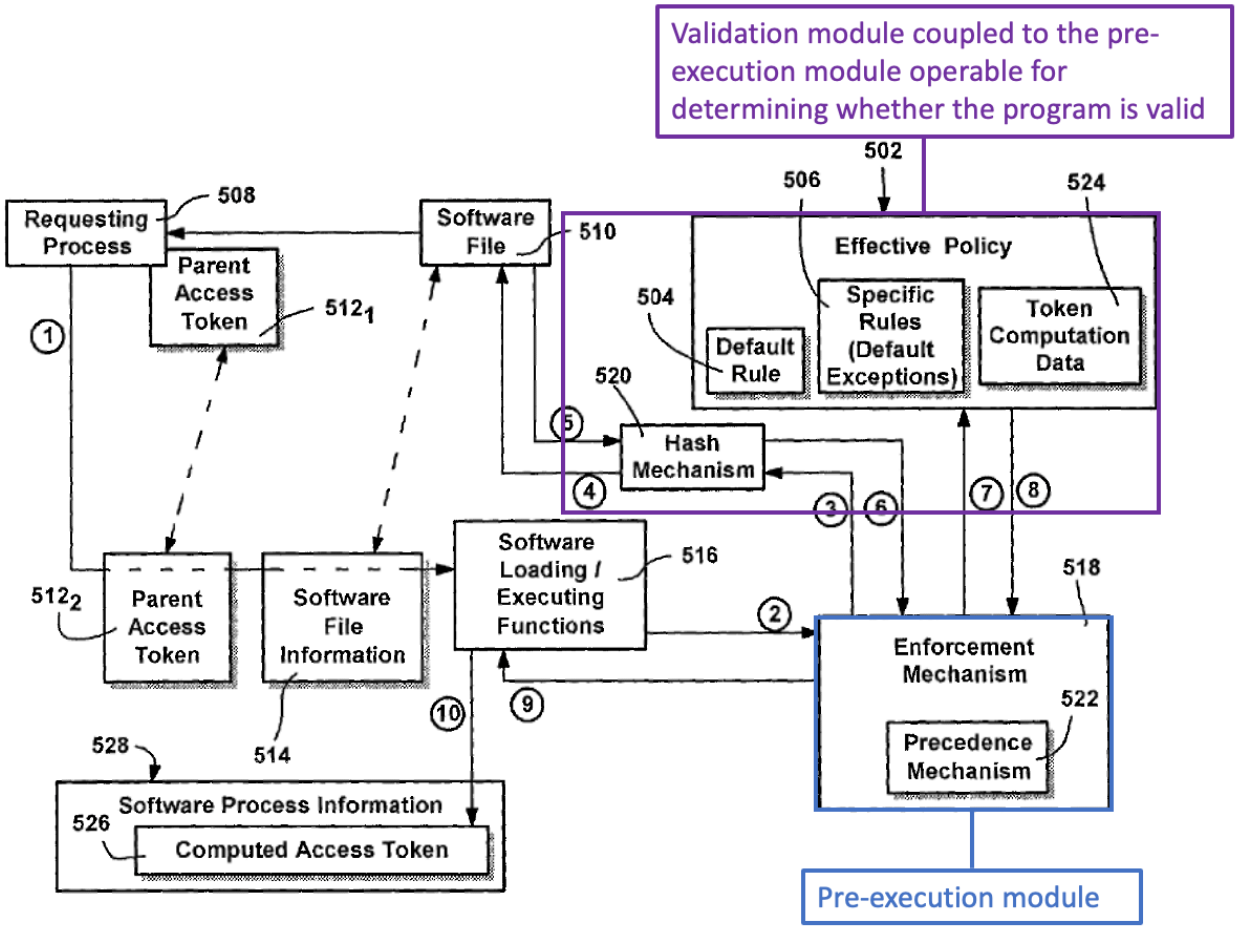
enforcement mechanism 518 (i.e., *the pre-execution module*) that is operable for determining whether a software file is valid. *Lambert*, Fig. 5A. After the enforcement mechanism 518 receives the notification from the software loading/executing functions 516, it "call[s]…a hash function 520 or the like to obtain a hash value from the contents of the software file 510, as generally represented in FIG. 5A by the arrows labeled with circled numerals three (3) to six (6)." *Id.*, [0068]. "[A] suitable hash function 520 provides a unique fingerprint of the software file 510, while the effective policy 502 may have a specific rule for that hash." *Id*. The hash is "a unique hash value of the file's content (e.g., via an MD5 image hash)." *Id.*, [0053].

To determine if the software file's hash matches the hash of known software, "the set of maintained hash values can be queried to determine whether a rule exists for it." *Id.,* [0055]. "[B]efore opening a file for execution, a hash value is determined from that file's contents, and the set of hash value information (e.g., in the policy object) [is] searched to determine if a rule exists for that hash value." *Id.* The effective policy 502 includes the set of hash value information for known software files and their associated rules. "[T]he effective policy 502 generally comprises the general (default) rule 504 and a set of specific rules 506 which set forth the exceptions to the default rule 502." *Id.*, [0065]. "Note that while FIG. 5A essentially represents accessing the policy to get the rule or rules via arrows labeled seven (7)

and eight (8),…e.g., to look first for a rule for the hash value[.]" *Id.*, [0068].

The rules stored in the policy 502 dictate whether the software file is valid. Specifically, "an administrator can select any software file, obtain a hash value from its contents, and enter and associate a rule (that specifies a security level) with that hash value." *Id.*, [0055]. "Multiple files can have their corresponding hash values and rules maintained in **a policy object**…whereby for any given instance of any file, the set of maintained hash values can be queried to determine whether a rule exists for it." *Id.* "In this manner, an administrator can set a rule for any file based on its unique hash value, with the rule specifying whether (and if so, how) a software file having that hash value will execute." *Id.*, [0056]. "In the case of known bad code, the administrator may set the rule to specify that any file with that hash will not be allowed to execute at all (i.e., 'disallowed' or 'revoked' status)." *Id.* "[I]n in the case of known good code that is allowed to execute, a hash rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent)." *Id.*

Thus, Lambert's effective policy 502 and hash mechanism 520 collectively qualify as *a validation module coupled to the pre-execution module operable for determining whether the program is valid*:



Declaration, ¶181.

[1(c)] a detection module coupled to the pre-execution monitor operable for intercepting a trigger from the computing device's operating system; and

Petitioner construes *the pre-execution monitor* as *the pre-execution module*. *See* Section II.C.2. Lambert teaches an instance where the application is allowed to execute in a restricted environment. *Lambert*, [0008]. A restricted execution environment "reduces software's access to resources, and/or removes privileges, relative to a user's normal access token." *Id.*, Abstract. When it is determined that software should be restricted, "the enforcement mechanism 518 can compute and

associate a restricted token with the software file 510, whereby if creation of a process is requested for the software file, (it includes executable content), any process of the software file is restricted in its access rights and privileges according to the restricted token." *Id.*, [0070]. "If the computed token 526 is a restricted token, the process of the software file can now execute, but only in association with the restricted token, thus providing the rule-determined secure execution environment." *Id.*

In Fig. 3, an untrusted process 300 having a restricted token 210 attempts to access object 302. *Id.*, Fig. 3, [0042].[6] "When the process 300 desires access to an object 302, the process 300 specifies the type of access it desires (e.g., obtain read/write access to a file object), and the operating system (e.g., kernel) level provides its associated token to an object manager 304." *Id.* Thus, Lambert's system presumes that an untrusted process will notify the object manager 304 when attempting to access a file object 302. However, a PHOSITA would have understood

[6] A PHOITA would have understood that the disclosures of Figure 3 and Figure 5A are not different embodiments because Figure 5A depicts pre-execution processes used to determine whether a program must execute in restricted mode via a restricted token and Figure 3 describes how a restricted program executes in conjunction with a restricted token. *Declaration*, ¶185.

that an untrusted process may attempt to bypass the object manager and directly access the operating system kernel to access the file object 302. *Declaration*, ¶187.

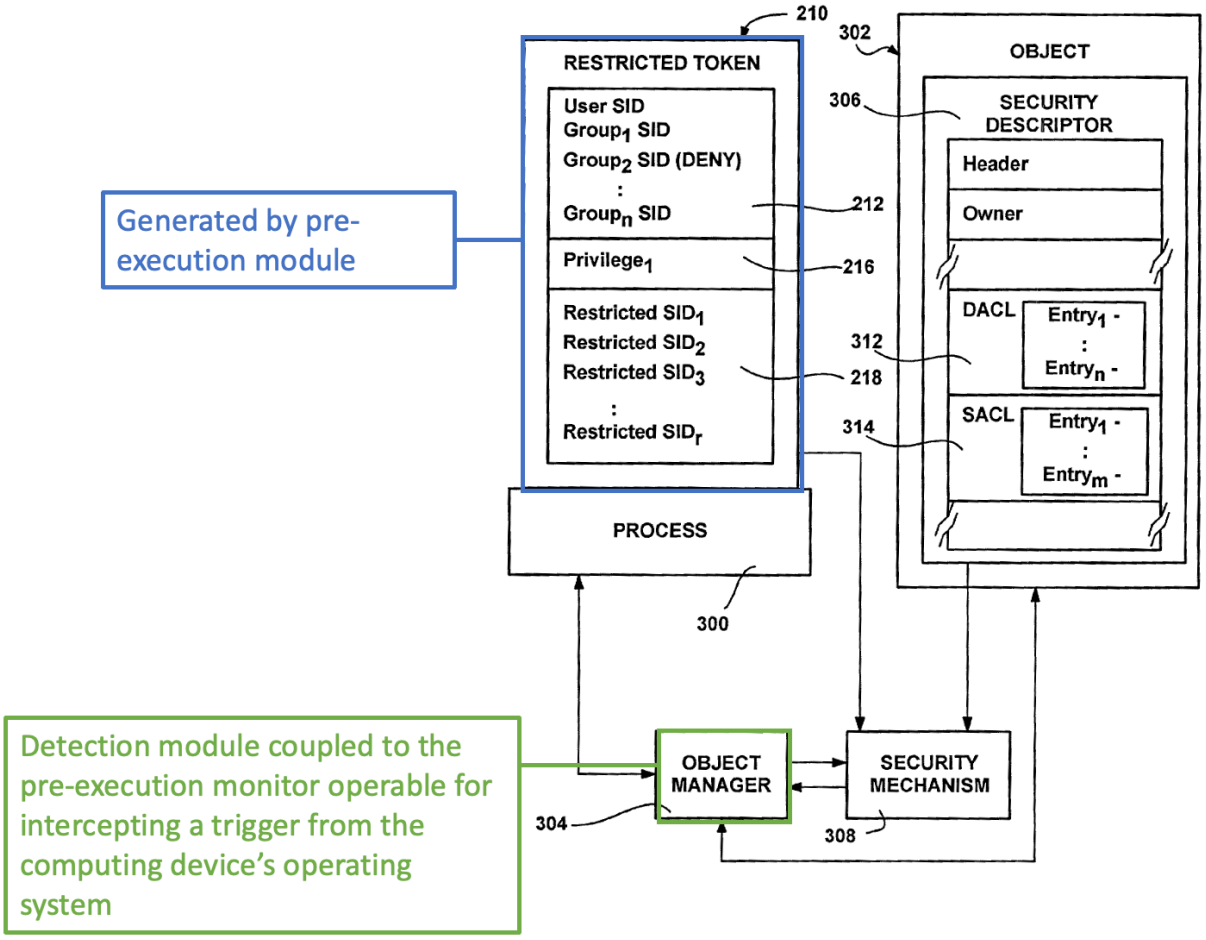
Schmid recognizes that "While most well-behaved Windows applications make system calls through the Win32 API, it is also possible to bypass this interface by making calls directly to kernel services." *Schmid*, 5:29-32. "System services are operations performed by an operating system kernel on behalf of applications or other kernel components." *Id.*, 5:49-51. "For instance, manipulating CPU hardware, starting and stopping processes, and manipulating files are sensitive operations generally implemented by services at the kernel level." *Id.*, 5:53-56. Given Lambert's system can also be implemented in a Windows environment, a PHOSITA would have understood that an untrusted and potentially malicious software process may bypass Lambert's object manager 304 and access the file object 302 directly via the Windows operating system kernel. *Lambert*, [0037]; *Declaration*, ¶187.

To prevent an untrusted application from bypassing protections, Schmid teaches a device driver that is used "[t]o implement a non-bypassable kernel wrapper" that "intercept[s] system service requests." *Schmid*, 5:40-48. Schmid's device driver intercepts triggers from the computer's operating system. Specifically, Schmid teaches that in a Windows environment, "user applications invoke system services by executing an interrupt instruction." *Id.*, 5:57-59. "A kernel entity, known as a dispatcher, initially responds to an interrupt instruction, determines the nature

of an interrupt, and calls a function to handle the request." *Id.*, 5:62-64. "Two tables in kernel memory describe locations and parameter requirements of all functions available to a dispatcher," including a table that "specifies handlers for user requests." *Id.*, 5:63-65. Schmid teaches "constructing a device driver that may be dynamically loaded into a kernel, or may be loaded as part of a boot sequence." *Id.*, 6:6-9. "When a device driver loads, it modifies an entry in a table consulted by a dispatcher to handle an interrupt instruction." *Id.*, 6:9-11. "[T]he modified entry may cause a dispatcher to call an inserted function" instead of the intended operating system function. *Id.*, 6:11-15. Schmid further teaches "that the driver modifies only tables relevant to requests from user applications." *Id.*, 6:15-16. Thus, a PHOSITA would understand that Schmid's device driver intercepts a trigger from the operating system's dispatcher by modifying entries in the table consulted by the dispatcher to handle an interrupt instruction. *Declaration*, ¶188. As such, Schmid's device driver qualifies as *a detection module operable for intercepting a trigger from the computing device's operating system*. *Id.*

Lambert only provides a functional description of the object manager 304 and does not provide details as to how it is to be implemented. *Id.*, ¶189. For example, Lambert does not describe whether the object manager is implemented in user mode or kernel mode or how the object manager is notified when the untrusted program attempts to access an object 302. *Id.* Based on the teachings of Schmid, it would

have been obvious, and a PHOSITA would have been motivated to implement Lambert's object manager using Schmid's device driver (i.e., *a detection module*). *Id.* Specifically, it would have been obvious to configure Lambert's system to implement the object manager by loading a device driver that modifies entries in a table consulted by the operating system's dispatcher and inserts corresponding functions that access the untrusted process's restricted token and the object 302's security descriptor. *Id*. A PHOSITA would have understood that this would have ensured that Lambert's object manager is notified when the untrusted process attempts to access an object 302 by making calls either via the Win32 API or via the operating system kernel. *Id*. Since the object manager accesses the untrusted process's restricted token and the restricted token is generated by the enforcement mechanism (i.e., *the pre-execution module*), the object manager is coupled to the enforcement mechanism. *Id.* Thus, the combination of Lambert and Schmid teaches *a detection module coupled to the pre-execution monitor operable for intercepting a trigger from the computing device's operating system*. *Id.*



Id.

Given the above-described advantages of implementing Lambert's object manager such that it cannot be bypassed by a malicious program, a PHOSITA would have been motivated to implement Lambert's object manager using Schmid's device driver to improve similar computer security systems in the same way. *Id.*, ¶190. This modification would have ensured that Lambert's object manager would be aware when an untrusted program is attempting to access resources it is restricted from accessing. *Id.* Thus, this modification would have predictably enhanced the

robustness of Lambert's computer security system by ensuring that a malicious program is not able to bypass its protections by making calls directly to the operating system kernel. *Id.*

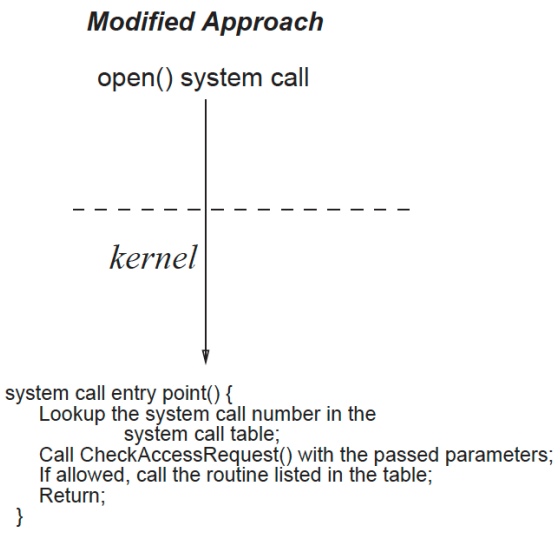
Writing a device driver, such as described by Schmid, was a well-known and predictable way to inject custom code into the kernel space of a Windows computing device. *Id.*, ¶191, ¶51. Likewise, the operation of the Windows operating system, including the operation of the dispatcher, was well documented and understood by PHOSITAs. *Id.,* ¶191. For these reasons, there would have been a reasonable expectation of success implementing Lambert's object manager using Schmid's device driver. *Id.*

1[(d)] an execution module coupled to the detection module and operable for monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module.

Lambert teaches a security mechanism 308 that receives the untrusted process's restricted token 210 and object 302's security descriptor 306 from the object manager 304 (i.e., <u>*the detection module*</u>). *Lambert*, [0043]. "The security mechanism 308 compares the security IDs in the restricted token 210 along with the type of action or actions requested by the process 300 against the entries in the DACL [discretionary access control list] 312," where the DACL includes "access control entries, and for each entry, one or more access rights (allowed or denied actions) corresponding to that entry." *Id.*, [0043]. "If a match is found with an

allowed user or group, and the type of access desired is allowable for the user or group, a handle to the object 302 is returned to the process 300, otherwise access is denied." *Id.* Again, Lambert provides a functional description of the security mechanism but does not provide implementation details, such as whether the security mechanism is implemented in user mode or kernel mode.

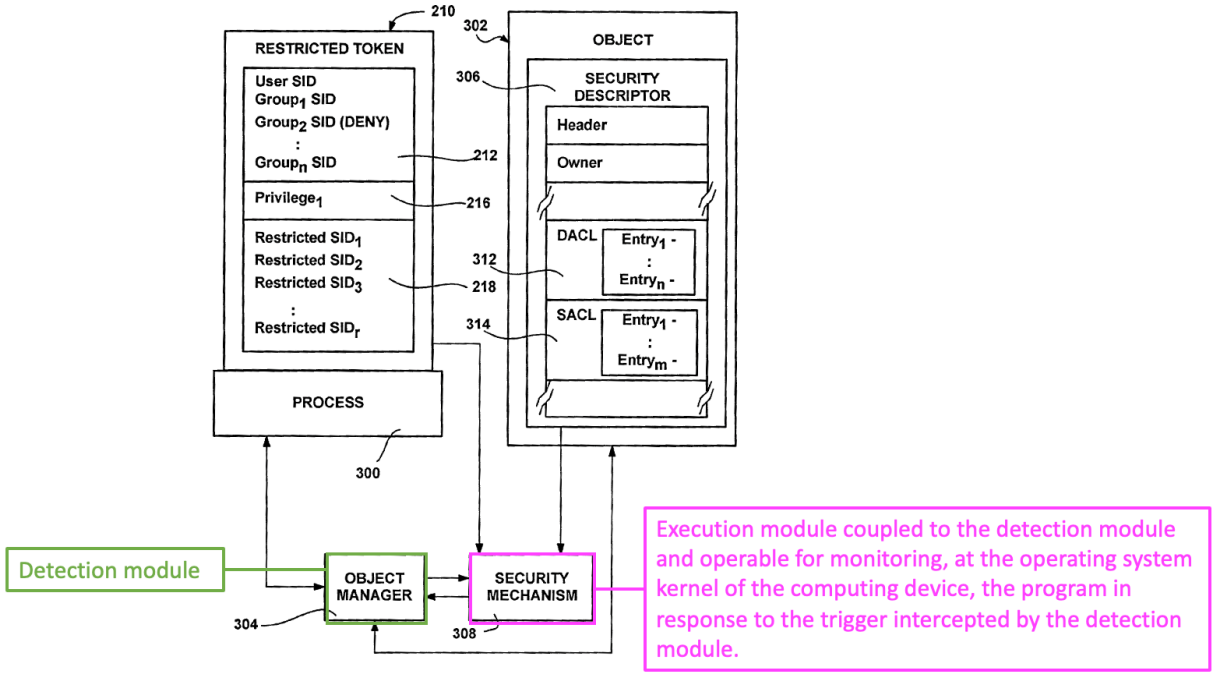
Dan teaches a similar monitoring system, including an RCL enforcer that "ensures that the permissions and resource consumption limits specified in the RCL are not violated." *Dan*, 9. To accomplish this, Dan's "RCL enforcer monitors all systems calls, i.e., access to requests to logical resources in the system." *Id.*, 12. "This monitoring requires the RCL enforcer to be invoked for each system call made by an application." *Id.* "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT data structures and either allows or disallows the call from continuing." *Id.* Unlike Lambert, Dan does provide implementation details for the RCL enforcer. Specifically, Dan teaches a system call modification approach where "the operating system's call interfaces is modified to call the CheckAccessRequest() upon entry to verify the parameters of the system call." *Id.*, 14. As shown in the figure below, this monitoring occurs on the operating system kernel:



Id., 15. Dan prefers this approach "because of the consistency it offers for enforcing security," and it "will ensure consistent security checks for all applications that are executed on the machine." *Id.*, 14-15.

The combination of Lambert and Schmid teaches that the object manager 304 is implemented via a device driver loaded into the operating system kernel. *See* Claim 1(c). Since Lambert's security mechanism works in tandem with the object manager to monitor the untrusted process's behavior, it would have been obvious to a PHOSITA to implement the security mechanism at the computing device's operating system kernel level, as taught by Dan. *Declaration*, ¶194. Implementing the security mechanism in the kernel would have allowed it to monitor the operating system calls intercepted by the object manager and ensure that the untrusted process is in compliance with its restricted token. *Id.* Thus, the combination of Lambert, Schmid, and Dan teaches *an execution module coupled to the detection module and*

operable for monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module:



Id.

As discussed above, Petitioner interprets this limitation under 112(6) where the function is "monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module" and the corresponding structure is "if the program was not validated, then monitor the nonvalidated program in response to triggers while the program is executing." *See* Section II.C.3 For the reasons discussed above, the Lambert-Schmid-Dan combination teaches a security mechanism 308 that monitors an untrusted (i.e., not validated) program in response to triggers intercepted by the object manager while the untrusted program is executing. *Declaration*, ¶195. Additionally, this structure

performs the function of "monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module" for the same reasons. *Id.*

A PHOSITA would have understood that implementing Lambert's security monitor on the kernel level, as taught by Dan, would have allowed the object manager, which is also operating in the computing device's kernel space, to efficiently communicate with the security mechanism without requiring communication between kernel mode and user mode. *Id.*, ¶196. Moreover, a PHOSITA would have understood that maintaining the security mechanism in the kernel mode as opposed to the user mode would have decreased the probability of a malicious program or user attempting to disable or tamper with the security mechanism. *Id.* As taught by Dan, a PHOISTA would have recognized that implementing Lambert's security mechanism in the kernel would have offered increased consistency for enforcing security. *Id.* For these reasons, a PHOSITA would have been motivated to combine prior art elements according to known methods and implement Lambert's security mechanism in the operating system kernel so that it can monitor the program in response to triggers intercepted by the object manager. *Id.* Due to the increased security and the predictable art, there would have been a reasonable expectation of success implementing Lambert's security monitor in the operating system kernel. *Id.*

2. Claim 2

2. The system of claim 1, wherein the pre-execution module is further operable for suspending loading of the program onto the computing device.

As discussed with regard to Claim 25(c), Lambert's enforcement mechanism 518 (i.e., *the pre-execution module*) is further operable for suspending the loading of the program onto the computing device. *See* Section V.A.5 at Claim 25(c).

VI. DISCRETIONARY DENIAL IS NOT WARRANTED

There is no basis to discretionarily deny institution under §314(a), at least because there is no operative trial date set between the parties. Beginning on August 3, 2022, pursuant to 28 U.S.C. §1407(a), the United States Judicial Panel on Multidistrict Litigation ("MDL Panel") centralized several litigations from various districts involving PO in the Eastern District of Texas ("EDTX"), including the litigation involving the parties to this proceeding. Ex. 1011. The centralized proceedings were assigned to Judge Gilstrap for pretrial management only and must be statutorily remanded to the originating court before trial. 28 U.S.C. §1407(a) ("Each action so transferred shall be remanded by the panel at or before the conclusion of such pretrial proceedings to the district from which it was transferred unless it shall have been previously terminated"). While Judge Gilstrap has set trial for EDTX cases on April 15, 2024, no trial has been set in the originating district. Ex. 1012, 1 (noting the "trial date applies only to cases originally filed in [EDTX]"). Therefore, trial must be set by the eventual trial court pending that court's trial

calendar. Furthermore, the schedule in EDTX may also be extended. Petitioner also intends to file a motion to transfer the case to California once the case is remanded to the originating court, followed by a motion to stay the case at the originating court pending transfer determination and pending IPR, leaving a trial date even more uncertain.

Further, Petitioner stipulates that if instituted, Petitioner will not seek resolution in the District Court of any ground of invalidity for the '137 Patent that utilizes 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 '137 Patent invalidity theories before the Board and at issue in the District Court. Petitioner's proposed stipulation is far more restrictive than a *Sand Revolution* stipulation, which excludes only the same grounds advanced in the IPR, allowing the Petitioner to rely on the same references in the two tribunals so long as the reliance is not identical.

Therefore, with an unknown trial date and any overlap mitigated by stipulation, denial under § 314(a) is not warranted.

VII. CONCLUSION

For the forgoing reasons, Petitioner respectfully requests *inter partes* review of the Challenged Claims.

Respectfully submitted,
/s/ Adam P. Seitz

ERISE IP, P.A.
Adam P. Seitz, Reg. No. 52,206
COUNSEL FOR PETITIONER

VIII. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8

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

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

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

The '137 Patent has been asserted in the following patent infringement lawsuits:

- *In re: Taasera Licensing LLC, Patent Litigation*, 2:22-md-03042 (Judicial Panel On Multidistrict Litigation);
- *Taasera Licensing LLC v. CrowdStrike, Inc. et al.*, 6:22-cv-01094 (WDTX);
- *Taasera Licensing LLC v. CrowdStrike, Inc. et al.*, 2:22-cv-00468 (EDTX);
- *Taasera Licensing LLC v. Musarubra US LLC, dba Trellix*, 2:22-cv-00427 (EDTX);
- *Trend Micro Incorporated et al. v. Taasera Licensing LLC*, 3:22-cv-00477 (NDTX);
- *Trend Micro Incorporated v. Taasera Licensing LLC*, 3:22-cv-00518 (NDTX);
- *Trend Micro Inc v. Taasera Licensing LLC*, 2:22-cv-00303 (EXTX);

- *Taasera Licensing LLC v. Trend Micro Incorporated*, 2:21-cv-00441 (EDTX);
- *Palo Alto Networks, Inc. v. Taasera Licensing LLC, et al.*, 2:22-cv-0314 (EDTX);
- *Palo Alto Networks, Inc. v. Taasera Licensing LLC, et al.*, 1:22-02306 (SDNY);
- *Taasera Licensing LLC v. Palo Alto Networks, Inc.*, 2:22-cv-00062 (EDTX); and
- *Taasera Licensing LLC v. Palo Alto Networks, Inc.*, 2:23-cv-00113 (EDTX).

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

Petitioner provides the following designation and service information for lead and backup counsel. 37 C.F.R. § 42.8(b)(3) and (b)(4).

Lead Counsel	**Back-Up Counsel**
Adam P. Seitz (Reg. No. 52,206) Adam.Seitz@eriseip.com PTAB@eriseip.com Postal and Hand-Delivery Address: ERISE IP, P.A. 7015 College Blvd., Suite 700 Overland Park, Kansas 66211 Telephone: (913) 777-5600 Fax: (913) 777-5601	Paul R. Hart, (Reg. No. 59,646) Paul.Hart@eriseip.com PTAB@eriseip.com Postal and Hand-Delivery Address: ERISE IP, P.A. 717 17th Street, Suite 1400 Denver, Colorado 80202 Telephone: (913) 777-5600 Fax: (913) 777-5601

APPENDIX OF EXHIBITS

1001	U.S. Patent No. 7,673,137
1002	File History of U.S. Patent No. 7,673,137
1003	Declaration of Markus Jakobsson, Ph.D.
1004	Declaration of June Munford
1005	Asit Dan, Ajay Mohindra, Rajiv Ramaswami and Dinkar Sitaram, *ChakraVyuja (CV): A Sandbox Operating System Environment for Controlled Execution of Alien Code*, IBM Research Division (February 20, 1997)
1006	U.S. Patent Publication No. 2002/0099952 to Lambert et al.
1007	U.S. Patent No. 7,085,928 to Schmid et al.
1008	Declaration of Dr. Eric Cole Regarding Claim Construction dated July 13, 2023
1009	Taasera Licensing LLC's Opening Claim Construction Brief
1010	Defendants' Joint Claim Construction Brief
1011	United States Judicial Panel on Multidistrict Litigation Transfer Order dated August 3, 2022
1012	Eighth Amended Docket Control Order
1013	Webster's New World Dictionary of Computer Terms, Simon & Schuster, Inc. (5th ed., 1994)
1014	*Intel Architecture Software Developer's Manual Volume 3: System Programming*, Intel Corporation (1999)
1015	David A. Solomon, *The Windows NT Kernel Architecture*, 31 Computer 40-47 (October 1998)
1016	Carol Neshevich, *NT 5.0 becomes Windows 2000*, 8 New World Canada 6, https://www.proquest.com/openview/fa3551c04de4b0e4a14ccee74eb12f0f/1?pq-origsite=gscholar&cbl=43820 (November 20, 1998)
1017	*Microsoft Releases Windows 2000 to Manufacturing*, Microsoft Corporation (December 15, 1999) https://news.microsoft.com/1999/12/15/microsoft-releases-windows-2000-to-manufacturing/
1018	Terrance Mitchem, Raymond Lu, Richard O'Brien, Kent Larson, *Linux Kernel Loadable Wrappers*, in Proceedings DARPA Information Survivability Conference and Exposition (IEEE, January 2000)
1019	Microsoft Press Computer Dictionary, Microsoft Corporation (3rd ed., 1997)

1020	Gary Wiggins, *Living with MalWare*, Security Essentials version 1.2d, (SANS Institute, 2001)
1021	Saliman Manap, *Rootkit: Attacker undercover tools*., Global Information Assurance Certification Paper, (10/21/2001)
1022	Massimo Bernaschi, Emanuele Gabrielli, Luigi V. Mancini, *Operating System Enhancement to Prevent Misuse of System Calls*, ACM CCS'00 (2000)
1023	Pietro Iglio, *TrustedBox: a Kernel-Level Integrity Checker*, in Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99) (IEEE, December 1999)
1024	F. De Paoli, A. L. Dos Santos, R. A. Kemmerer, *Vulnerability of "Secure" Web Browsers*, In Proceedings of the National Information Systems Security Conference (May 26, 1997)
1025	Bart Preneel, *Cryptographic Hash Functions*, 5 European Transactions on Telecommunications (1994)
1026	karlrupp, *Microprocessor Trend Data*, GitHub, https://github.com/karlrupp/microprocessor-trend-data#readme (Feb 22, 2022)
1027	*Why is Sandboxing important?,* STL Tech (October 14, 2022)

CERTIFICATION OF WORD COUNT

The undersigned certifies pursuant to 37 C.F.R. §42.24 that the foregoing Petition for *Inter Partes* Review, excluding any table of contents, mandatory notices under 37 C.F.R. §42.8, certificates of service or word count, or appendix of exhibits, contains 13,927 words according to the word-processing program used to prepare this document (Microsoft Word).

Dated: October 24, 2023

Respectfully submitted,

BY: */s/ Adam P. Seitz*

Adam P. Seitz, Reg. No. 52,206
Adam.Seitz@eriseip.com
Erise IP, P.A.
7015 College Blvd., Suite 700
Overland Park, KS 66211
P: (913) 777-5600
F: (913) 777-5601

COUNSEL FOR PETITIONER

CERTIFICATE OF SERVICE ON PATENT OWNER
UNDER 37 C.F.R. § 42.105(a)

Pursuant to 37 C.F.R. §§ 42.6(e) and 42.105(b), the undersigned certifies that on October 24, 2023, a complete and entire copy of this Petition for *Inter Partes* Review and Exhibits were provided via Federal Express to the Patent Owner by serving the correspondence address of record for the '137 Patent:

International Business Machines Corporation
Intellectual Property Law Department (Maintenance)
T.J. Watson Research Center
1101 Kitchawan Road, Route 134, P.O. Box 218
YORKTOWN, NY 10598
UNITED STATES

Petitioner has also served copies of this Petition on Patent Owner's litigation counsel via e-mail:

Alfred Ross Fabricant
ffabricant@fabricantllp.com;

Joseph Michael Mercadante
jmercadante@fabricantllp.com;

Vincent J Rubino , III
vrubino@fabricantllp.com;

Peter Lambrianakos
plambrianakos@fabricantllp.com;
Fabricant LLP
411 Theodore Fremd Avenue
Suite 206 South
Rye, NY 10580
212-257-5797
Fax: 212-257-5796

Julian Glenn Pymento
jpymento@fabricantllp.com
Fabricant LLP

51 JFK Parkway
Short Hills, NJ 07078
646-797-4341

Jennifer Leigh Truelove
jtruelove@mckoolsmith.com;

Samuel Franklin Baxter
sbaxter@mckoolsmith.com
McKool Smith, P.C. - Marshall
104 East Houston St
Suite 300
Marshall, TX 75670
903-923-9000
Fax: 903-923-9099

Justin Kurt Truelove
kurt@truelovelawfirm.com
Truelove Law Firm
207 N Wellington
P O Box 1409
Marshall, Tx 75671
903-938-8321
Fax: 903-215-8510

Raymond W Mort, III
raymort@austinlaw.com;
The Mort Law Firm, PLLC
501 Congress Ave, Suite 150
Austin, TX 78701
512-865-7950
Fax: 512-865-7950

Respectfully submitted,

BY: */s/ Adam P. Seitz*
Adam P. Seitz, Reg. No. 52,206
Adam.Seitz@eriseip.com
Erise IP, P.A.

7015 College Blvd., Suite 700
Overland Park, KS 66211
P: (913) 777-5600
F: (913) 777-5601

COUNSEL FOR PETITIONER