Trials@uspto.gov
Tel: 571-272-7822

Paper 9
Entered: January 14, 2016

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SYMANTEC CORP.,
Petitioner,

v.

FINJAN, INC.,
Patent Owner.

Case IPR2015-01552
Patent 7,757,289 B2

Before THOMAS L. GIANNETTI, RICHARD E. RICE, and MIRIAM L. QUINN, *Administrative Patent Judges.*

RICE, *Administrative Patent Judge*.

DECISION
Denying Institution of *Inter Partes* Review
37 C.F.R. § 42.108

I. INTRODUCTION

Petitioner Symantec Corporation filed a Petition (Paper 1, "Pet.") requesting an *inter partes* review of claims 10–12, 15, 17, 19–24, 35, 36, 38, 39, 41, 42, 44, and 45 ("the challenged claims") of U.S. Patent No. 7,757,289 B2 (Ex. 1001, "the '289 Patent"). Patent Owner Finjan Inc. filed

a Preliminary Response (Paper 8, "Prelim. Resp."). We have jurisdiction under 35 U.S.C. § 314, which provides that an *inter partes* review may not be instituted "unless . . . there is a reasonable likelihood that the petitioner would prevail with respect to at least 1 of the claims challenged in the petition." 35 U.S.C. § 314(a). As Petitioner has not shown a reasonable likelihood that it would prevail with respect to at least one of the challenged claims, we do not institute an *inter partes* review with respect to the '289 Patent.

A. Related Proceedings

We are informed that Petitioner is named as a defendant in a federal district court case involving the '289 Patent (*Finjan, Inc. v. Symantec Corp.*, Case No. 3:14-cv-02998-RS (N.D. CA)). Pet. 1. We also are informed that Petitioner has filed petitions requesting *inter partes* review of U.S. Patent Nos. 8,141,154 (IPR2015-01547); 8,015,182 (IPR2015-01548); 7,930,299 (IPR2015-01549), and 7,756,996 (IPR2015-01545 and IPR2015-01546). *See id.*

B. The '289 Patent

The '289 Patent, titled "System and Method for Inspecting Dynamically Generated Executable Code," issued July 13, 2010 from U.S. Application No. 11/298,475, filed December 12, 2005. Ex. 1001, at (54), (45), (21), (22). The '289 Patent seeks to solve problems posed by dynamically generated viruses that "are themselves generated only at run-time." *Id.* at 3:31–35, 4:37–40. According to the Specification, a solution to this problem involves using a gateway computer to inspect incoming network content and to replace "original function calls" with "substitute function calls" that enable the client computer to pass "function inputs" to a

security computer at run-time and to suspend processing of content pending replies from the security computer. *Id.* at 4:64–5:2. The security computer shields the client computer while the network content is being processed:

> During run-time, while processing the network content, but before the client computer invokes a function call that may potentially dynamically generate malicious code, the client computer passes the input to the function to the security computer for inspection, and suspends processing the network content pending a reply back from the security computer. Since the input to the function is being passed at run-time, it has already been dynamically generated and is thus readily inspected by a content inspector.

Id. at 4:44–53.

The ’289 Patent describes a system that includes gateway computer 205 (including content modifier 265), client computer 210 (including content processor 270), and security computer 215 (including input inspector 275 and input modifier 285). *Id.* at 9:16–22. Figure 2 of the ’289 Patent, which depicts this system, is reproduced below.

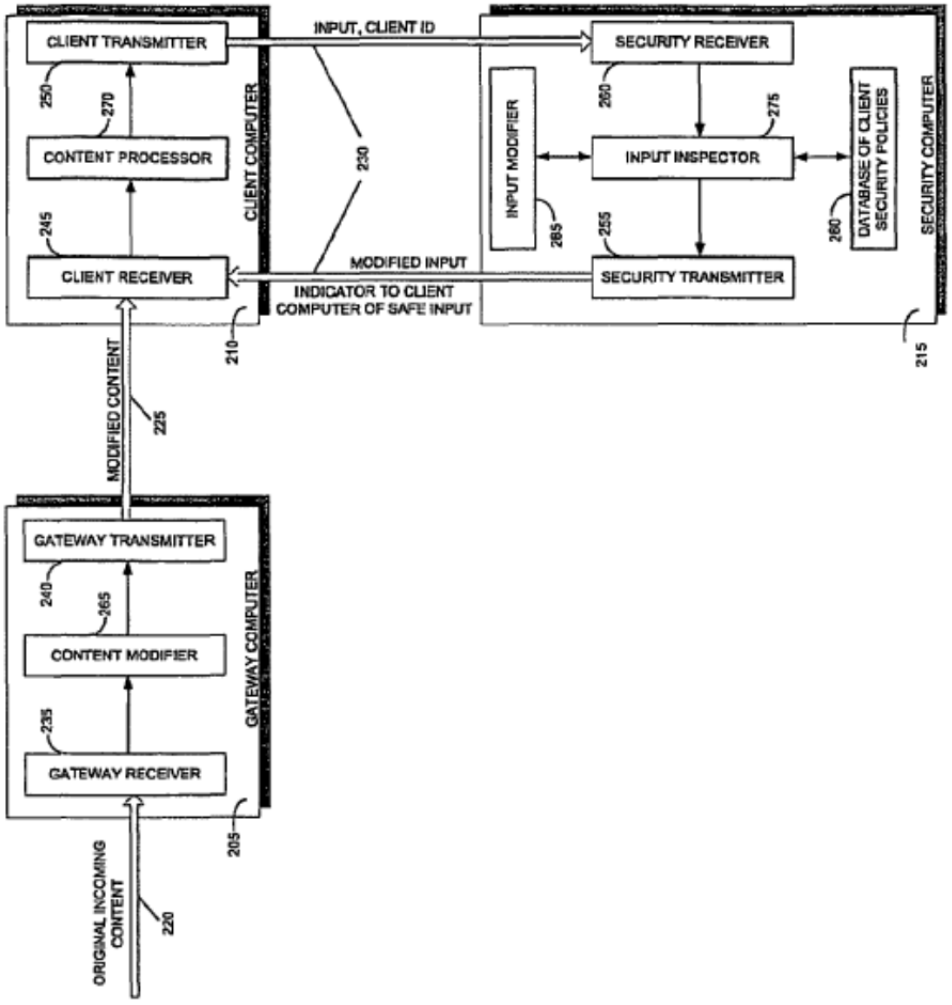

FIG. 2

Figure 2 is a simplified block diagram that illustrates a preferred embodiment for protecting client computer 210 from dynamically generated malicious executable code. *Id.* at 8:51–54. In the system depicted in Figure 2, content modifier 265 of the gateway computer scans the original content, identifies function calls, and replaces the original function calls with substitute function calls. *Id.* at 9:18–50. Content modifier 265 identifies function calls of the form

Function(input). (1)

Id. at 9:26–30. Content modifier 265 modifies function calls (1) to corresponding substitute function calls of the form

```
Substitute_function(input,*),                    (2)
```

"whereby the call to Function( ) has been replaced with a call to Substitute_function( ) . . . [and] the input intended for the original function is also passed to the substitute function, along with possible additional input denoted by "*". *Id.* at 9:31–38.

Content processor 270 of the client computer processes the modified content. *Id.* at 11:15–16. When content processor 270 invokes the substitute function call, the function input is passed to security computer 215 for inspection. *Id.* at 11:15–19. Until security computer 215 returns its inspection results to the client computer, processing of the modified content is suspended. *Id.* at 11:19–21.

A specific problem that the '289 Patent seeks to solve involves "recursive levels of dynamic generation of malicious code, whereby such code is generated via a series of successive function calls, one within the next." *Id.* at 5:4–6. As an example, the Specification describes function call (5), which involves two levels of function:

```
Document.write("<h1>Document.write(                    (5)
    "<h1><SCRIPT>Some JavaScript</SCRIPT></h1>")</h1>") .
```

Id. at 12:58–60. According to the Specification, function call (5) "first calls Document.write( ) to generate the function call (3)," which is reproduced below:

```
Document.write("<h1><SCRIPT>Some    JavaScript
    </SCRIPT></h1>").                    (3)
```

Id. at 12:7–9, 62–64.[1] As further described in the Specification, function call (5) "then calls Document.write() again to generate the JavaScript." *Id.* at 12:63–64. The problem posed by these two levels of function calls, as stated in the Specification, is that: "If the inputs to each of the Document.write() invocations in (5) are themselves dynamically generated at run-time, then one pass through [the] input inspector may not detect the JavaScript." *Id.* at 12:64–67.

To solve this problem, input inspector 275 preferably passes inputs it receives to input modifier 285, prior to scanning the input. *Id.* at 13:1–2. "Input modifier [285] preferably operates similar to content modifier 265, and replaces function calls detected in the input with corresponding substitute function calls." *Id.* at 13:3–5.

In the example above, when client computer 210 invokes the outer call to Document.write() in (5), input test string (6), which is reproduced below, is passed to security computer 215:

"<hl>Document.write((6)
"<hl>SCRIPT>Some JavaScript</SCRIPT></hl>")</hl>"

Id. at 13:5–13. Input modifier 285 detects the inner function call to Document.write(), replaces it with a corresponding substitute function call, and returns the modified input to client computer 210. *Id.* at 13:14–16. Although input inspector 275 may not have detected the presence of the

[1] Function call (3) "serves to instruct content processor 270 to insert the text between the <hl >header tags into the HTML pages; namely the text <SCRIPT> JavaScript</SCRIPT> which itself invokes the JavaScript between the <SCRIPT>tags." *Id.* at 12:10–14.

JavaScript in the first inspection, content processor 270 will invoke the substitute function for Document.write() when it begins to process the modified content on resumption of processing; and the substitute function will pass the input of the inner Document.write() call of (5) to security computer 215 for inspection. *Id.* at 13:17–29. "This time around input inspector 275 is able to detect the presence of the JavaScript, and can analyze it accordingly." *Id.* at 13:29–31.

C. Illustrative Claim

Claims 10, 19, 22, 35, and 41 are independent. Claims 11, 12, 15, and 17 depend directly from claim 10; claims 20 and 21 depend directly from claim 19; claims 23 and 24 depend directly from claim 22; claims 36, 38, and 39 depend directly from claim 35; and claims 42, 44 and 45 depend directly from claim 41. Claim 10 is illustrative of the claimed subject matter, and is reproduced below:

> 10. [Pre] A system for protecting a computer from dynamically generated malicious content, comprising:
> [A] a gateway computer, comprising:
> [B] a gateway receiver for receiving content being sent to a client computer for processing, the content including a call to an original function, and the call including an input;
> [C] a content modifier for modifying the received content by replacing the call to the original function with a corresponding call to a substitute function,
> [D] the substitute function being operational to send the input to a security computer for inspection; and
> [E] a gateway transmitter for transmitting the modified content from the gateway computer to the client computer;
> [F] the security computer, comprising:
> [G] a security receiver for receiving the input from the client computer;

> [H] an input modifier for modifying the input if the input itself includes a call to a second original function with a second input by replacing the call to the second original function with a corresponding call to a second substitute function,
>
> [I] the second substitute function being operational to send the second input to the security computer for inspections;
>
> [J] an input inspector for determining whether it is safe for the client computer to invoke the original function; and
>
> [K] a security transmitter for transmitting the modified input to the client computer, if the input was modified by said input modifier, and for
>
> [L] transmitting an indicator of the determining to the client computer; and
>
> [M] a client computer communicating with said gateway computer and with said security computer, comprising:
>
> [N] a client receiver for receiving the modified content from said gateway computer, for
>
> [O] receiving the modified input, if the input was modified by said input modifier, and for
>
> [P] receiving the indicator from said security computer;
>
> [Q] a content processor for processing the modified content, and for
>
> [R] invoking the original function only if the indicator indicates that such invocation is safe; and
>
> [S] a client transmitter for transmitting the input to said security computer for inspection, when the substitute function is invoked.

Id. at 18:59–19:36 (emphasis added); *see* Pet. 8–10 (designating the recitations of claim 10 "[Pre]" and "[A]" through "[S]," as shown above)

D. The Asserted Grounds

Petitioner challenges claims 10–12, 15, 17, 19–24, 35, 36, 38, 39, 41, 42, 44, and 45 of the '289 Patent on the following grounds (Pet. 4):

References	Basis	Claims Challenged
Calder[2] and Sirer[3]	§ 103(b)	10–12, 15, 17, 19–24, 35, 36, 38, 39, 41, 42, 44, and 45
Ross[4] and Calder	§ 103(a)	10–12, 15, 17, 19–24, 35, 36, 38, 39, 41, 42, 44, and 45

In addition to Calder, Sirer, and Ross, Petitioner relies on the Declaration of Jack W. Davidson, Ph.D. (Ex. 1009).

II. ANALYSIS

We turn now to Petitioner's asserted grounds of unpatentability to determine whether Petitioner has met the threshold standard of 35 U.S.C. § 314(a) for instituting review.

A. Claim Construction

In an *inter partes* review, the Board gives claim terms in an unexpired patent their broadest reasonable interpretation in light of the specification of the patent in which they appear. 37 C.F.R. § 42.100(b); *see also In re*

[2] U.S. Patent Application Publication No. 2002/0066022 A1 (Ex. 1003), published May 30, 2002.

[3] Emin Gün Sirer et al., *Design and implementation of a distributed virtual machine for networked computers*, Association of Computing Machinery (December 1999) (Ex. 1004). Petitioner relies on the Declaration of Sylvia Hall-Ellis (Ex. 1005) and Exhibits 1007 and 1008 to establish that Sirer is a printed publication that was publicly available by February 7, 2000. Pet. 3–4.

[4] U.S. Patent Application Publication No. 2007/0113282 A1 (Ex. 1002), filed November 17, 2005, and published May 17, 2007. Petitioner asserts that Ross is prior art to the challenged claims "under Pre-AIA 35 U.S.C. §102(e)." Pet. 3.

Cuozzo Speed Techs., LLC, 793 F.3d 1268, 1278, 1279 (Fed. Cir. 2015). Under the broadest reasonable interpretation standard, and absent any special definition, claim terms are given their ordinary and customary meaning, as would be understood by one of ordinary skill in the art in the context of the entire disclosure. *In re Translogic Tech., Inc.*, 504 F.3d 1249, 1257 (Fed. Cir. 2007). Any special definition for a claim term must be set forth with reasonable clarity, deliberateness, and precision. *In re Paulsen*, 30 F.3d 1475, 1480 (Fed. Cir. 1994).

In this case, Petitioner proposes an express claim construction for the term "dynamically generated." *Id.* at 11–12. Patent Owner opposes Petitioner's proposed claim construction, but itself does not propose an express construction. Prelim. Resp. 7–8. We do not resolve this claim construction dispute between the parties, however, because none of our determinations regarding Petitioner's proposed grounds of unpatentability requires us to interpret expressly the term "dynamically generated" or any other claim term.

B. Asserted Obviousness

1. Legal Principles; Level of Skill in the Art

In an *inter partes* review, obviousness must be based on prior art consisting of patents or printed publications. 35 U.S.C. § 311(b). A claim is unpatentable for obviousness under 35 U.S.C. § 103(a) if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art ("POSA") to which the subject matter pertains. *See KSR Int'l Co. v. Teleflex Inc.*, 550 U.S. 398, 406 (2007). A patent claim composed of several elements,

however, is not proved obvious merely by demonstrating that each of its elements was known, independently, in the prior art. *Id.* at 418. In analyzing the obviousness of a combination of prior art elements, it can be important to identify a reason that would have prompted one of skill in the art to combine the elements in the way the claimed invention does. *Id.* A precise teaching directed to the specific subject matter of a challenged claim is not necessary to establish obviousness. *Id.* Rather, "any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed." *Id.* at 420. The question of obviousness is resolved on the basis of underlying factual determinations, including: (1) the scope and content of the prior art; (2) any differences between the claimed subject matter and the prior art; (3) the level of skill in the art; and (4) objective evidence of nonobviousness, i.e., secondary considerations, when in evidence. *Graham v. John Deere Co.*, 383 U.S. 1, 17–18 (1966).

Here, Petitioner defines the level of skill in the art as follows:

> A person of ordinary skill in the art ("POSITA") . . . at the time of the alleged invention of the '289 patent would generally have a master's degree in computer science, computer engineering, or a similar [field], or a bachelor's degree in computer science, computer engineering, or a similar field, with approximately two years of experience in the fields of networking and anti-malware development, computer security or equivalent work experience. Additional graduate education might substitute for experience, while significant experience in the field of computer programming, networking, and/or malicious code might substitute for formal education.

Pet. 11 (Ex. 1009 ¶ 29). Patent Owner does not dispute Petitioner's definition of the level of skill in the art, with which we agree, and we adopt it for purposes of our Decision.

2. Calder and Sirer

In arguing that that claims 10–12, 15, 17, 19–24, 35, 36, 38, 39, 41, 42, 44, and 45 of the '289 Patent would have been obvious over Calder and Sirer, Petitioner asserts that independent claims 19, 22, 35, and 41 are each directed to a subset of the claim limitations found in independent claim 10, including limitation [H] (identified *supra* in Section I.C). *See id.* at 7–10, 16–39. Below, we focus our obviousness analysis on limitation [H] of claim 10, which pertinently recites "modifying the input if the input itself includes a call to a second original function with a second input by replacing the call to the second original function with a corresponding call to a second substitute function."

a. Overview of Calder and Sirer

As characterized by Petitioner, "Calder teaches a pre-processor module that generates a modified application binary[5] by scanning program code for system calls (original functions) and rewriting the program code to trap the call to an interception module (a substitute function) instead." Pet. 12 (citing Ex. 1003, Abstract, Fig. 2). The interception module, Petitioner asserts, is the entry point to a virtual machine that provides virtual interfaces (e.g., filesystem, network,

[5] Calder uses interchangeably the terms "application binary" and "application program." Ex. 1003 ¶ 76.

registry) to the client for executing the modified application. *Id.* at 12–13 (citing Ex. 1003 ¶¶ 88, 77, 85, Fig. 4; Ex. 1009 ¶¶ 68–72). By initially intercepting part or all of the application interface ("API") routines, the interception module can prevent an application program from improperly modifying or accessing data from the client computer. *Id.* at 13–14 (citing Ex. 1003 ¶¶ 86–87; Ex. 1009 ¶ 74).

Petitioner contends that, to the extent "a separate, remotely located 'security computer' for performing the inspection of the hooked functions and inputs" is not disclosed by Calder, this feature is disclosed by Sirer. *Id.* at 14 (citing Ex. 1009 ¶¶ 75–78). As characterized by Petitioner, "Sirer describes a distributed virtual machine (DVM) architecture where 'system services, such as verification, *security enforcement*, compilation and optimization, *are factored out of clients and located on powerful network servers*.'" *Id.* (citing Ex. 1004, Abstract). According to Petitioner, "Sirer teaches using a security service to 'check user-supplied arguments to system calls.'" *Id.* (citing Ex. 1004, 3).

b. Petitioner's Arguments

Petitioner contends that Calder discloses limitation [H] of claim 10. Pet. 25–28.[6] According to Petitioner, Calder discloses "initially replacing an application's system calls with substitute calls to an interception module [and] . . . also explains that a potentially malicious application may attempt to incorporate code dynamically (*i.e.*, code that is not subject to the initial check)." *Id.* at 25. Petitioner further asserts:

[6] Petitioner does not rely on Sirer for limitation [H] of claim 10. *See* Pet. 25–28.

> To address this, Calder teaches that the same replacement process is performed on any code that is dynamically generated by the application. More specifically, Calder teaches "scanning the dynamically generated code, that is created by the application, for code sequences that cause the computer to trap to the operating system, and means for modifying the code sequences."

Id. (citing Ex. 1003 ¶¶ 15, 19; Ex. 1009 ¶ 105).

Petitioner states that "this second level of scanning and replacement occurs during runtime when the application attempts to make a memory page executable as part of an intercepted 'modify page permissions' call." *Id.* (citing Ex. 1003 ¶ 198, Fig. 13). Figure 33 of Calder is reproduced below.

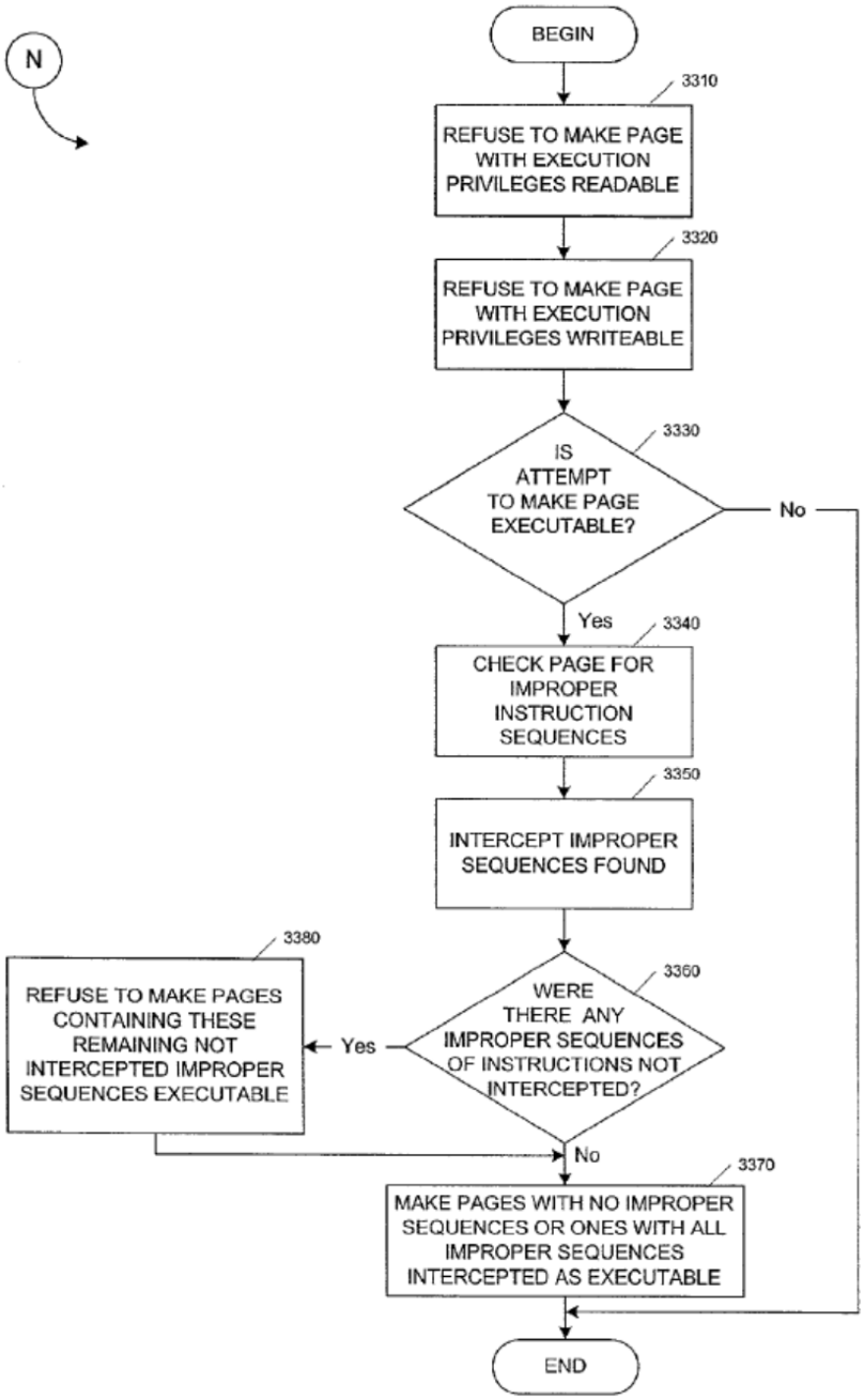
N
BEGIN
3310
REFUSE TO MAKE PAGE WITH EXECUTION PRIVILEGES READABLE
3320
REFUSE TO MAKE PAGE WITH EXECUTION PRIVILEGES WRITEABLE
3330
IS ATTEMPT TO MAKE PAGE EXECUTABLE?
No
Yes
3340
CHECK PAGE FOR IMPROPER INSTRUCTION SEQUENCES
3350
INTERCEPT IMPROPER SEQUENCES FOUND
3360
WERE THERE ANY IMPROPER SEQUENCES OF INSTRUCTIONS NOT INTERCEPTED?
Yes
3380
REFUSE TO MAKE PAGES CONTAINING THESE REMAINING NOT INTERCEPTED IMPROPER SEQUENCES EXECUTABLE
No
3370
MAKE PAGES WITH NO IMPROPER SEQUENCES OR ONES WITH ALL IMPROPER SEQUENCES INTERCEPTED AS EXECUTABLE
END

FIG. 33

Figure 33 of Calder "is a flowchart illustrating a process for intercepting and virtualizing a modify page permissions routine [i.e., call] that was invoked by the application 405." Ex. 1003 ¶ 198.

Petitioner argues that "inputs to this call would include the memory page to be modified and the permissions." Pet. 25–26 (citing Ex. 1009 ¶ 106). Petitioner further argues:

> Once the modify page permission call is intercepted, Calder teaches that "it is determined whether the application is requesting to make the pages executable." . . .
>
> Calder explains that when an application package attempts to make a memory page executable, the original hooking and interception process is performed again in order to prevent the application from jumping to the new page and making improper system calls directly. In this case, the input "pages are checked for improper sequences. Progressing to step 3350, the improper sequences are rewritten to be intercepted, *i.e.*, rewritten to call the interception routine."

Id. at 26 (citing Ex. 1003 ¶¶ 199, 200); *see* Fig. 33. Petitioner reads limitation [H] of claim 10 on the process illustrated in Figure 33 of Calder as follows:

> Accordingly Calder discloses that the interception module (i.e ., input modifier) modifies the input (*e.g.*, a memory page) which includes a call to a second original function (*e.g.*, another system call), by replacing it with a call to a second substitute function (*i.e.*, a call back to the interception module).

Id. (citing Ex. 1009 ¶¶ 108–109).

Alternatively, Petitioner asserts that "Calder also describes similar techniques for rewriting DLL's that are loaded during program execution." *Id.* Petitioner explains that, "[l]ike the memory pages, a DLL may not be a part of the application binary but, rather, only accessed/loaded during run-

time, when instructed by the application." *Id.* at 26–27 (citing Ex. 1003 ¶ 98). Petitioner further explains that, because a malicious application could use an un-modified DLL to make improper system calls, Calder teaches initially loading a DLL for the interception module before any other DLL. *Id.* at 27 (citing Ex. 1009 ¶ 110). Petitioner states:

> Upon execution, the interception module is loaded first and when the application attempts to load another DLL, Calder teaches that "all DLL routines that are to be intercepted are redirected to a wrapper routine to intercept them. The interception module DLL performs its API patching for every DLL that has been loaded."

Id. (citing Ex. 1003 ¶ 105, Figs. 10, 11). Petitioner reads limitation [H] of claim 10 on Calder's interception of DLL routines as follows:

> Calder discloses that the interception module (*i.e.*, input modifier) modifies the input variable (*e.g.*, a DLL to be loaded) which includes a call to a second original function (*e.g.*, API call), by replacing it with a call to a second substitute function (*i.e.*, a call back to the interception module).

Id. at 27–28 (citing Ex. 1009 ¶ 112).

c. Analysis

Limitation [H] of claim 10 requires modifying "*the input*" if the input itself includes a call to a second original function with a second input by replacing the call to the second original function with a corresponding call to a second substitute function. The recitation of "the input" in limitation [H] refers back to "an input" in limitation [B], which recites "a gateway receiver for receiving content being sent to a client computer for processing, the content including a call to an original function, and the call including *an*

input" (emphasis added). *Thus, limitation [H] requires modifying the input to an original function that includes a call to a second original function.*

We are not persuaded by Petitioner's arguments that Calder discloses modifying the input to an original function that includes a call to a second original function, as required by limitation [H] of claim 10. *See* Prelim. Resp. 15–19. We agree that Calder discloses: (1) scanning an application for code sequences that cause the computer to trap to the operating system, and modifying the code sequences; and (2) scanning the dynamically generated code that is created by the application for code sequences that cause the computer to trap to the operating system, and modifying the code sequences. *See* Pet. 25 (citing Ex. 1003 ¶¶ 15, 19); Ex. 1009 ¶ 105. Petitioner has not explained sufficiently, however, why modifying the code sequences of such dynamically generated code, as disclosed by Calder, corresponds to modifying the input to an original function that includes a call to a second original function, as the claim requires.

In particular, Petitioner has not persuaded us that Calder's process for intercepting and virtualizing a modify page permissions routine satisfies this claim requirement. *See* Ex. 1003 ¶¶ 199–200, Fig. 33. Calder discloses that, "[a]s part of invoking the modify page permissions routine, the application identifies certain pages." Ex. 1003 ¶ 199, Fig. 33. If the application requests to make the pages executable, the pages are checked for improper sequences, and the improper sequences are rewritten to call the interception routine. *Id.* ¶ 200, Fig. 33. Petitioner does not identify where Calder discloses that "inputs to [the modify page permissions call] would include *the memory page to be modified* and the permissions," as argued in the Petition. Pet. 25–26, emphasis added, citing Ex. 1009 ¶ 106). *See* Ex. 1003

¶¶ 198–200, Fig. 33; *see also* Prelim. Resp. 16 (arguing "that Petitioner cites to no evidence from Calder to support its position that a memory page could be an input to a function"). Likewise, Dr. Davidson does not identify in his Declaration where Calder discloses that the memory page to be modified is a function input, but rather merely reiterates Petitioner's conclusory argument. Ex. 1009 ¶ 106.

Petitioner similarly has not persuaded us that Calder's techniques for rewriting DLLs that are loaded during program execution involve modifying the input to an original function that includes a call to a second original function, as required by limitation [H] of claim 10. *See* Pet. 26–28. As disclosed in Calder, rewriting DLLs is part of the process for initializing an application and patching the loaded libraries. Ex. 1003 ¶ 103. The process involves loading into memory the libraries defined by the import tables of the application, executing the initialization routine of the first DLL in the import table (i.e., the DLL for the interception module[7]), and then patching the loaded libraries. *Id.* ¶¶ 104, 105. In the process of patching the loaded libraries, "all DLL routines that are to be intercepted are redirected to a wrapper routine to intercept them," and "[t]he interception module DLL performs its API patching for every DLL that has been loaded." *Id.* ¶ 105, Figs. 9, 10. Petitioner does not identify where Calder discloses that "a DLL to be loaded" is a function input, as argued in the Petition. *See* Pet. 27–28

[7] Calder discloses inserting a DLL for the interception module into an import table that lists all of the DLLs used by an application, such that the interception module DLL is invoked prior to the other DLLs. Ex. 1003 ¶ 98. "[S]ince the interception module is loaded and run first, the interception module can patch and intercept all of the DLL calls before any of the application package's code (including DllMain() routines) are executed." *Id.*

(citing Ex. 1009 ¶ 112); *see also* Prelim. Resp. 17 (arguing that "Calder simply does not disclose an input variable including a call to an additional function"). The cited paragraph of Dr. Davidson's Declaration, moreover, provides no information beyond reiterating Petitioner's conclusory argument. *See* Ex. 1009 ¶ 112.

For these reasons, we determine that Petitioner has *not* demonstrated a reasonable likelihood of prevailing with respect to its challenge to claim 10 and dependent claims 11, 12, 15, and 17 as obvious over Calder and Sirer. As independent claims 19, 22, 35, and 41, like independent claim 10, each require limitation [H] (*see* Pet. 9), we also determine that Petitioner has *not* demonstrated a reasonable likelihood of prevailing with respect to its challenge to claims 19–24, 35, 36, 38, 39, 41, 42, 44, and 45 as obvious over Calder and Sirer.

3. Ross and Calder

With respect to its challenge to claims 10–12, 15, 17, 19–24, 35, 36, 38, 39, 41, 42, 44, and 45 of the '289 Patent as obvious over Ross and Calder, Petitioner again argues that Calder discloses limitation [H] of independent claims 10, 19, 22, 35, and 41.[8] *Id.* at 48–49. For the reasons discussed *supra* in connection with the combination of Calder and Sirer, we are not persuaded that Calder discloses limitation [H]. Accordingly, we determine that Petitioner has *not* demonstrated a reasonable likelihood of prevailing with respect to its challenge to claims 10–12, 15, 17, 19–24, 35,

[8] Petitioner does not rely on Ross for limitation [H]. *See* Pet. 48–49.

36, 38, 39, 41, 42, 44, and 45 of the ’289 Patent as obvious over Ross and Calder.

III. CONCLUSION

For the foregoing reasons, we determine that Petitioner has not established a reasonable likelihood of prevailing on its challenges to claims 10–12, 15, 17, 19–24, 35, 36, 38, 39, 41, 42, 44, and 45 of the ’289 Patent.

IV. ORDER

In consideration of the foregoing, it is hereby:

ORDERED that Petitioner’s Petition for an *inter partes* review of claims 10–12, 15, 17, 19–24, 35, 36, 38, 39, 41, 42, 44, and 45 of U.S. Patent No. 7,757,289 B2 as obvious over (i) Calder and Sirer and (ii) Ross and Calder is *denied*, and no *inter partes* review will be instituted pursuant to 35 U.S.C. § 314 as to any claim of that patent on any of the grounds of unpatentability alleged by Petitioner in the Petition.

PETITIONER:

Joseph J. Richetti
Daniel A. Crowe
BRYAN CAVE LLP
joe.richetti@bryancave.com
dacrowe@bryancave.com

PATENT OWNER:

James Hannah
Jeffrey H. Price
KRAMER LEVIN NAFTALIS & FRANKEL LLP
jhannah@kramerlevin.com
jprice@kramerlevin.com

Michael Kim
FINJAN INC.
mkim@finjan.com