UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AO KASPERSKY LAB,
Petitioner

v.

OPEN TEXT INC.,
Patent Owner.

IPR2023-00895
Patent 10,284,591 B2

Before BARBARA A. PARVIS, MICHELLE N. WORMMEESTER, and AARON W. MOORE, *Administrative Patent Judges.*

MOORE, *Administrative Patent Judge*.

DECISION
Denying Institution of *Inter Partes* Review
35 U.S.C. § 314

I. INTRODUCTION

A. Background

AO Kaspersky Lab ("Petitioner") filed a Petition requesting *inter partes* review of claims 1–17 of U.S. Patent No. 10,284,591 B2 (Ex. 1001, "the '591 patent"). Paper 1 ("Pet."). Open Text Inc. ("Patent Owner") filed a Preliminary Response. Paper 6 ("Prelim. Resp.").

We may institute an *inter partes* review if "the information presented in the petition . . . and any response . . . shows that there is a reasonable likelihood that the petitioner would prevail with respect to at least 1 of the claims challenged in the petition." 35 U.S.C. § 314(a) (2018).

For the reasons provided below, we determine that Petitioner has not demonstrated a reasonable likelihood that it will prevail in showing the unpatentability of at least one challenged claim. Accordingly, we do not institute *inter partes* review.

B. Related Proceedings

The parties identify the following district court cases as related to this proceeding:

> *Webroot, Inc. et al. v. Trend Micro Inc.*,
> Case No. 6:22-cv-00239 (W.D. Tex.);
>
> *Webroot, Inc. et al. v. Sophos Ltd.*,
> Case No. 6:22-cv-00240 (W.D. Tex.);
>
> *Webroot, Inc. et al. v. CrowdStrike, Inc. et al.*,
> Case No. 6:22-cv-00241 (W.D. Tex.);
>
> *Webroot, Inc. et al. v. AO Kaspersky Lab*,
> Case No. 6:22-cv-00243 (W.D. Tex.); and
>
> *Webroot, Inc. v. Forcepoint LLC*,
> Case No. 6:22-cv-00342 (W.D. Tex.).

See Pet. 1–2; Paper 3, 1.

We previously instituted an *inter parties* review of the '591 patent in IPR2022-01522 and denied institution of an *inter partes* review of the '591 patent in IPR2023-00692. Both of those proceedings were based on references different than those presented by Petitioner.

C. The '591 Patent

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

Figure 3 of the patent is reproduced below:

FIG. 3

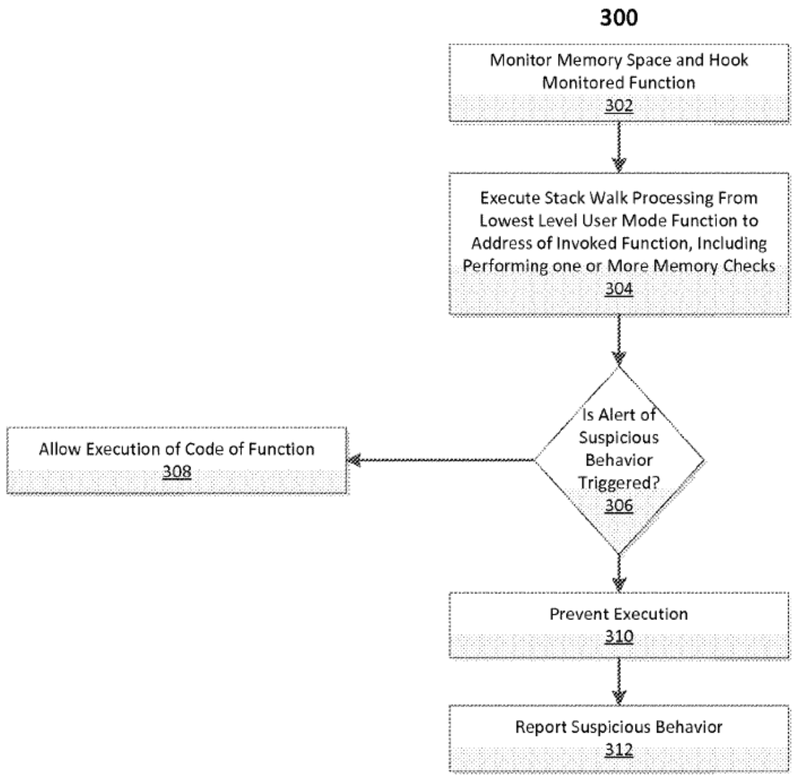
300
Monitor Memory Space and Hook Monitored Function 302
Execute Stack Walk Processing From Lowest Level User Mode Function to Address of Invoked Function, Including Performing one or More Memory Checks 304
Is Alert of Suspicious Behavior Triggered? 306
Allow Execution of Code of Function 308
Prevent Execution 310
Report Suspicious Behavior 312

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

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

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

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

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

> code execution is attempted from non-executable memory;
>
> a base pointer of a stack frame is invalid;
>
> a stack return address is invalid;
>
> attempted execution of a return-oriented programming technique is detected;
>
> the base pointer of a stack frame is outside a current thread stack; and/or
>
> a return address is inside a virtual memory area.

Id. at 8:18–33.

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

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

> 1. A computer-implemented method comprising:
>
> monitoring a memory space of a process for execution of at least one monitored function of a plurality of functions, wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space;
>
> invoking one of the plurality of functions as a result of receiving a call from an application programming instance;
>
> executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and

> performing, during the executing of the stack walk processing before an address of an originating caller function is reached, a memory check for a plurality of stack entries identified during the stack walk processing to detect suspicious behavior, wherein an alert of suspicious behavior is triggered when the performing of the memory check detects at least one of the following:
>
> code execution is attempted from non-executable memory,
>
> a base pointer is identified as being invalid,
>
> an invalid stack return address is identified,
>
> attempted execution of a return-oriented programming technique is detected,
>
> the base pointer is detected as being outside a current thread stack, and
>
> a return address is detected as being inside a virtual memory area,
>
> wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function from operating.

Ex. 1001, 13:29–61.

D. Asserted Grounds of Unpatentability

Petitioner's challenge raises the following grounds (Pet. 15):

Claims Challenged	Basis 35 U.S.C. §	Reference(s)
1, 3, 4, 6, 9, 11, 12, 14, 17	102/103	Fratric[1]
2, 5, 7, 8, 10, 13, 15, 16	103	Fratric, Sallam[2]

[1] Ivan Fratric, *Runtime Prevention of Return-Oriented Programming Attacks*, available at https://github.com/ivanfratric/ropguard/blob/master/doc/ropguard.pdf (Exhibit 1007).

[2] U.S. Patent App. Pub. No. 2012/0255018 (Exhibit 1008).

Petitioner also relies on the Declaration of Venkatramanan Siva Subrahmanian, filed as Exhibit 1002. Patent Owner relies on the Declaration of Nenad Medvidovic, Ph.D., filed as Exhibit 2001.

II. ANALYSIS

Fratric details a method much like that described and claimed in the ’591 patent. A component (“ropguarddll.dll” in Fratric’s example) is loaded into a memory space, which it monitors for the execution of a monitored function (called a “critical function” in Fratric). *See* Ex. 1007, 7. “[O]nce the dll has been injected into the target process, it . . . proceeds to inline-patch all of the critical functions defined in [a] configuration file to perform appropriate ROP checks when called.” Ex. 1007, 8. Then, a call from an application programming instance (e.g., that of Internet Explorer in the example) that invokes a monitored, patched function causes “stack frame checking,” which walks through the stack and checks the return addresses of the critical function and of the function that called the critical function. That appears to correspond to the claimed “invalid stack return address” check.[3] *See id.* at 5. Fratric explains that the system “prevent[s]” return-oriented programming attacks (*see id.* at 3), which Petitioner asserts is sufficient to disclose preventing execution of a payload for the function from operating when an alert of suspicious behavior is triggered, as claimed. *See* Pet. 36. Fratric explains that “[i]f a ROP attempt is discovered . . . a message box with the details about the problem will be displayed and the user can choose to terminate the process or continue the execution.” Ex. 1007, 8.

[3] The stack frame checking may be employed if the program is “compiled in such a way that it uses EBP register as a stack frame pointer.” Ex. 1007, 5.

Petitioner argues that "Fratric teaches every element of claims 1, 3, 4, 6, 9, 11, 12, 14, and 17," but that if Fratric does not disclose performing memory checks "before reaching the address of an originating caller function," that would have been obvious. *See* Pet. 24–47. Petitioner adds Sallam for features recited in dependent claims 2, 5, 7, 8, 10, 13, 15, and 16. *See id.* at 48–61. In view of the parties' arguments, we will focus our discussion on independent claim 1.

Regarding the content of the prior art, Patent Owner argues that Fratric does not disclose "monitoring a memory space . . . for execution of at least one monitored function," as recited in claim 1. *See* Prelim. Resp. 30–32. We do not agree because, as explained above, Fratric describes how ropguarddll.dll monitors the memory space for calls to execute critical functions. Patent Owner also argues that Fratric does not disclose claim 1's "invoking one of the plurality of functions as a result of receiving a call from an application programming instance." We disagree with that argument as well, because Fratric describes how an application (e.g., Internet Explorer) would invoke one of the patched critical functions, causing the stack frame checking. *See* Ex. 1007, 8. We thus find Patent Owner's arguments regarding the content of the prior art unpersuasive.

However, Patent Owner also argues that Petitioner has not shown Fratric to be a printed publication under 35 U.S.C. § 102. *See* Prelim. Resp. 22–29. We agree.

"In an IPR, the petitioner bears the burden of establishing by a preponderance of the evidence that a particular document is a printed publication." *Nobel Biocare Servs. AG v. Instradent USA, Inc.*, 903 F.3d 1365, 1375 (Fed. Cir. 2018) (as amended Sept. 20, 2018). "[T]he key inquiry is whether or not a reference has been made 'publicly accessible.'" *In re Klopfenstein*, 380 F.3d 1345, 1348 (Fed. Cir. 2004); *see Acceleration Bay, LLC v. Activision Blizzard Inc.*, 908 F.3d 765, 772 (Fed. Cir. 2018) (explaining that public accessibility is "the touch-stone" in determining whether a reference qualifies as a printed publication).

Petitioner argues that Fratric qualifies as a printed publication because it "was published at least by August 26, 2012, to GitHub, an Internet hosting service for software development and version control that is commonly used to host opensource and other public software projects." Pet. 16 (citing Ex. 1002 ¶¶ 51–52). For support, the Petition includes an annotated screenshot from GitHub, depicting a portion of the first page of Fratric and the GitHub header, which shows the Fratric filename ("ropguard.pdf"), a "commit" date of August 26, 2012, and a repository visibility of "Public." *See* Pet. 17.

We agree with Patent Owner that the screenshot is insufficient to establish that Fratric was a printed publication prior to the January 27, 2015 filing date of the '591 patent. First, there is nothing in the record to show that the screenshot was of material that existed prior to the '591 patent's filing date, as there is no citation for the screenshot, it does not appear to be taken from an exhibit, and it is undated.[4] Second, the record shows that the visibility setting of a GitHub repository may be changed (*see* Ex. 1011), and

[4] Patent Owner provides a portion of a screenshot from the "Wayback Machine" indicating that https://github.com/ivanfratric/ropguard was not indexed earlier than June 11, 2018. *See* Prelim. Resp. 25. This does not prove the repository was not available earlier (it may have existed earlier but just not have been crawled), but it does indicate that Petitioner's publication date cannot be corroborated by the Wayback Machine.

there is no evidence that the "Public" setting shown in the screenshot had not changed over time. This means that if the screenshot was taken after the priority date (which it almost certainly was), it is not sufficiently probative of the visibility status before the priority date.

Petitioner asserts that "Fratric was published at least as of the GitHub 'Latest commit' date of August 26, 2012, displayed on the publicly-accessible GitHub, Inc. webpage." Pet. 17 (citing Ex. 1002 ¶ 77). This is unpersuasive because although a "commit" records file additions or changes in a GitHub branch (*see* Ex. 1010), those new or changed files would not be accessible by the public if the repository was set to private and, as explained above, Petitioner has not shown that the Fratric repository was publicly visible prior to the critical date.

For these reasons, we conclude that Petitioner has not met its burden of establishing, by a preponderance of the evidence, that Fratric is a Section 102 printed publication. It follows that Petitioner has not established a reasonable likelihood of proving at least one claim of the '591 patent unpatentable.

III. CONCLUSION

Having determined that Petitioner has not established a reasonable likelihood of proving that at least one claim of the '591 patent is unpatentable over the cited prior art, we do not institute an *inter partes* review.[5]

[5] Because we deny the Petition on the merits, we do not reach Patent Owner's request that we exercise our discretion to deny the Petition based on the state of the related litigation. *See* Prelim. Resp. 3–20.

IV. ORDER

It is ORDERED that no *inter partes* review is instituted in this proceeding.

For PETITIONER:

Joseph T. Miotke
Eric Chadwick
Christian Girtz
Erin Block
DEWITT LLP
jtm@dewittllp.com
ehc@dewittllp.com
cgirtz@dewittlp.com
eblock@dewittllp.com

For PATENT OWNER:

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