

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

TREND MICRO, INC.
Petitioner

v.

WEBROOT INC.

Case No. IPR2023-00692
Patent No. 10,284,591

**PETITION FOR INTER PARTES REVIEW
OF U.S. PATENT NO. 10,284,591**

Filed on behalf of Trend Micro, Inc.
By: Robert M. Hansen
Matthew J. Anderson
Scott B. Amankwatia
The Marbury Law Group, PLLC
11800 Sunrise Valley Drive
15th Floor
Reston, VA 20191

Tel: (703) 391-2900
Fax: (703) 391-2901

TABLE OF CONTENTS

I. Mandatory Notices (37 C.F.R. § 42.8(a)(1)).....	1
PAYMENT OF FEES.....	3
II. Grounds for Standing (37 C.F.R. § 42.104(a))	3
III. Identification of Challenge (37 C.F.R. § 42.104(b))	3
A. Citation of Prior Art.....	3
B. Statutory Grounds for Challenge.....	4
IV. The ‘591 Patent.....	4
A. Claim of Priority of the ‘591 patent	4
B. Specification of the ‘591 Patent	5
C. Summary of the Prosecution History	8
D. Level of Ordinary Skill in the Art	9
E. Claim Construction.....	9
V. Grounds of Rejection	10
A. Ground 1: Claims 1-18 are unpatentable under 35 U.S.C. § 103(a) as obvious based on Dalcher in view of Pappas.....	10
1. Claim 1	14
a. A computer-implemented method comprising:	14
b. 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;	14
c. invoking one of the plurality of functions as a result of receiving a call from an application programming instance;	17
d. executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and	18

e.	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;	21
f.	wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function from operating.	25
2.	Claim 2	26
	The method according to claim 1, further comprising:	26
a.	identifying an address associated with the invoked function.....	26
b.	when the address of the invoked function address is found in a cache, preventing execution of a payload for the invoked function.....	27
3.	Claim 3	29
	The method according to claim 1, wherein executing of the stack walk processing further comprises walking a stack frame associated with the invoked function from a low level user mode function to the originating address of the invoked function.	29
4.	Claim 4	31
	The method according to claim 1, wherein the monitoring further comprises monitoring functions associated with memory management in the memory space by a user mode function.	31
5.	Claim 5	32
	The method according to claim 1, wherein the monitoring further comprises applying, upon detecting a function call of one of the plurality	

of functions in the monitored memory space, a hooking technique to hook a lowest level user mode function of the invoked function, wherein the executing of the stack walk processing executes a reverse stack walk from the lowest level user mode function of the invoked function.	32
6. Claim 6	34
The method according to claim 1, wherein the plurality of functions comprises an NtAllocateVirtualMemory function, an NtProtectVirtualMemory function, an NtCreateProcess function, an NtCreateProcessEX function, an NtCreateUserIProcess function, and an LdrLoadDll function.	34
7. Claim 7	35
The method according to claim 5, wherein performing the memory check comprises evaluating state information associated with a cache entry for the hooked function.....	35
8. Claim 8	36
The method according to claim 7, wherein when an address of the invoked function is found in a cache, preventing execution of a payload for the invoked function.....	36
9. Claim 9	38
A system comprising:	38
a. at least one memory; and.....	38
b. at least one processor connected with the memory configured to perform operation of:	38
c. 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;	39
d. invoking one of the plurality of functions as a result of receiving a call from an application programming instance;	42

e.	executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and	43
f.	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,	46
g.	wherein when an alert of suspicious behavior is triggered, the method further comprising preventing execution of a payload for the invoked function.	50
10.	Claim 10	51
	The system according to claim 9, further comprising identifying an address associated with the invoked function; and when the address of the invoked function is found in a cache, and wherein the processor further executing an operation of preventing execution of a payload for the invoked function.	51
11.	Claim 11	52
	The system according to claim 9, wherein the executing of the stack walk processing further comprises walking a stack frame associated with the invoked function from a low level user mode function to the originating address of the invoked function.	52
12.	Claim 12	52
	The system according to claim 9, wherein the monitoring further comprises monitoring functions associated with memory management in the memory space by a user mode function.	52

13.	Claim 13	53
	The system according to claim 9, wherein the monitoring further comprises applying, upon detecting a function call of one of the plurality of functions in the monitored memory space, a hooking technique to hook a lowest level user mode function of the invoked function, and wherein the executing of the stack walk processing executes a reverse stack walk from the lowest level user mode function of the invoked function.	53
14.	Claim 14	54
	The system according to claim 9, wherein the plurality of functions comprises an NtAllocateVirtualMemory function, an NtProtectVirtualMemory function, an NtCreateProcess function, an NtCreateProcessEX function, an NtCreateUserIProcess function, and an LdrLoadDll function.	54
15.	Claim 15	54
	The system according to claim 13, wherein performing of the memory check comprises evaluating state information associated with a cache entry for the hooked function.	54
16.	Claim 16	55
	The system according to claim 15, wherein when an address of the invoked function is found in a cache, and wherein the processor further executing an operation of preventing execution of a payload for the invoked function.	55
17.	Claim 17	56
	a. A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:	56
	b. 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;	56

c.	invoking one of the plurality of functions as a result of receiving a call from an application programming instance;	56
d.	executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and	57
e.	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, a return address is detected as being inside a virtual memory area, and a low level user mode function is attempted to be executed in a cache, wherein the cache is evaluated for an address of the invoked function,.....	58
f.	wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function.....	59
18.	Claim 18	60
a.	The computer-readable storage device according to claim 17, wherein performing of the memory check to determine if code execution is attempted from non-executable memory, the base pointer is identified as being invalid, an invalid stack return address is identified, execution of a return-oriented programming technique is attempted, the base pointer is detected as being outside a current thread stack, and the return address is detected as being inside a virtual memory area, occurs before an address of the invoked function is reached, and	60
b.	wherein the performing of the memory check to determine whether the low level user mode function is attempted to be executed in the cache occurs upon reaching the address of the invoked function in the execution of the stack walk processing.....	62

VI. NO BASIS FOR DISCRETIONARY DENIAL	64
VII. CONCLUSION	66

TABLE OF AUTHORITIES

Cases

<i>Advanced Bionics, LLC v. MED-EL Elektromedizinische Gerate GmbH</i> , IPR2019-01469, Paper 6 (Feb. 13, 2020).....	65
<i>Cellco Partnership v. Huawei Device Co.</i> , IPR2020-01117, Paper 10 (Feb. 3, 2021)	65
<i>General Electric v. Vestas</i> , IPR2018-00928, Paper 9, *16-17 (PTAB Nov. 5, 2018).....	10
<i>Graham v. John Deere Co.</i> , 383 U.S. 1 (1966)	10
<i>KSR Int’l Co. v. Teleflex, Inc.</i> , 550 U.S. 398 (2007)	10, 11
<i>Phillips v. AWH Corp.</i> , 415 F.3d 1303 (Fed. Cir. 2005) (en banc)	9
<i>Titanium Metals Corp. of Am. v. Banner</i> , 778 F.2d 775,782 (Fed. Cir. 1985)	25, 58
<i>Western Digital Corporation v. Spex Technologies, Inc.</i> , IPR2018-00084, Paper 14, *11 (PTAB Apr. 25, 2018)	10

Statutes

35 U.S.C. § 102(a)(1).....	3, 4
35 U.S.C. § 103	4

Rules

37 C.F.R. § 42.10(a).....	2
37 C.F.R. § 42.104(a).....	3
37 C.F.R. § 42.8(a)(1)	1
37 C.F.R. § 42.8(b)(2).....	1
37 C.F.R. § 42.8(b)(3).....	2

EXHIBIT LIST

Exhibit No.	Description
1001	U.S. Patent No. 10,284,591 to Giuliani <i>et al.</i> ('591 patent).
1002	Expert Declaration of Dr. Seth Nielson
1003	<i>Curriculum Vitae</i> of A.L Dr. Seth Nielson.
1004	U.S. Patent Publication No. 2013/0275981 to Dalcher.
1005	Vasilis Pappas, <i>Transparent ROP Exploit Mitigation Using Indirect Branch Tracing</i>, 22nd USENIX Security Symposium, (August 14-16, 2013).
1006	Prosecution History of the '591 patent.
1007	Provisional patent application claimed by '591 patent for priority.
1008	David A. Patterson, <i>Computer Organization & Design The Hardware/Software Interface</i>, Morgan Kaufmann Publishers, Inc., (1994).
1009	ORS Open Systems Resources, Inc., <i>Collecting Detailed Performance Data with Xperf</i>, The NT Insider 17(1), (2010).
1010	Aleph One, <i>Smashing the Stack for Fun and Profit</i>, (1996).
1011	Jacob R. Lorch, <i>The VTrace Tool: Building a System Tracer for Windows NT and Windows 2000</i>, MSDN Magazine, (2000)
1012	Marco Pistoia, <i>Beyond Stack Inspection: A Unified Access-Control and Information-Flow Security Model</i>, IEE, (2007).

1013	Galen Hunt, <i>Detours: Binary Interception of Win32 Functions</i>, Microsoft Research, (1999).
1014	Sandra Loosemore, <i>The GNU C Library Reference Manual</i>, 188-193, (1999).
1015	Jonathan Corbet, <i>Expanding the Kernel Stack</i>, LWN.net, (2014).
1016	OpenSSL documentation API listing, https://www.openssl.org/docs/man1.0.2/man3/.
1017	Declaration of Ingrid Hsieh-Yee, Ph.D.

Trend Micro, Inc. (“Petitioner”) petitions for *inter partes* review of claims 1-18 of U.S. Patent No. 10,284,591 to Giuliani *et al.*, entitled “DETECTING AND PREVENTING EXECUTION OF SOFTWARE EXPLOITS” (“the ’591 patent,” EX1001), which according to PTO records, is assigned to Webroot, Inc.

I. Mandatory Notices (37 C.F.R. § 42.8(a)(1))

REAL PARTY IN INTEREST:

The real parties-in-interest are Trend Micro, Inc., Trend Micro America, Inc. and Trend Micro Incorporated. Trend Micro, Inc. is a wholly owned subsidiary of Trend Micro America, Inc., which is a wholly owned subsidiary of Trend Micro Incorporated, a Japanese Corporation.

RELATED MATTERS:

Pursuant to 37 C.F.R. § 42.8(b)(2), Petitioner identifies the following related matters.

A petition for *inter partes* review of the ’591 patent was filed by CrowdStrike, Inc as IPR2022-01522 on September 30, 2022.

The ’591 patent is at issue in the following district court litigations:

- *Webroot, Inc. and Open Text, Inc. v. Trend Micro Inc.*, Case No. 6:22-cv-00239 (W.D. Tex.), which was originally filed on March 4, 2022.

- *Webroot, Inc. and Open Text, Inc. v. Sophos LTD.*, Case No. 6:22-cv-00240 (W.D. Tex.), which was originally filed on March 4, 2022.
- *Webroot, Inc. and Open Text, Inc. v. CrowdStrike, Inc. and CrowdStrike Holdings, Inc.*, Case No. 6:22-cv-00241 (W.D. Tex.), which was originally filed on March 4, 2022.
- *Webroot, Inc. and Open Text, Inc. v. AO Kaspersky Lab*, Case No. 6:22-cv-00243 (W.D. Tex.), which was originally filed on March 4, 2022.

LEAD AND BACKUP COUNSEL:

Pursuant to 37 C.F.R. § 42.8(b)(3) and 37 C.F.R. § 42.10(a), Petitioner appoints:

Robert M. Hansen (Reg. No. 43,656) as lead counsel,

Scott B. Amankwatia (Reg. No. 78,226) as back-up counsel.

Matthew J. Anderson (Reg. No. 78,257) as back-up counsel.

Pursuant to 37 C.F.R. § 42.10(b), a power of attorney is being filed with this designation of counsel.

SERVICE INFORMATION:

Petitioner provides the following service information for designated counsel:

The Marbury Law Group, PLLC
11800 Sunrise Valley Drive
15th Floor
Reston, VA 20191
Tel: (703)-391-2900
Fax: (703)-391-2900

Petitioner consents to electronic service by email to the email addresses:

rhansen@marburylaw.com;

samankwatia@marburylaw.com

manderson@marburylaw.com; and

pat-docketing@marburylaw.com.

PAYMENT OF FEES

The PTO is authorized to charge any fees due during this proceeding to Deposit Account No. 18-1579.

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

Petitioner certifies that the '591 patent is available for *inter partes* review and that Petitioner is not barred or estopped from requesting an *inter partes* review challenging the patent claims on the grounds identified in this Petition.

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

A. Citation of Prior Art

Petitioner cites the following prior art references:

U.S. Published Patent Application 20130275981A1 to Dalcher, (“Dalcher”, EX1004) is prior art under 35 USC § 102(a)(1) because it was published on October 17, 2013, before the earliest possible priority date of the '591 patent.

Dalcher was not submitted to or considered by the Examiner who allowed the '591 patent.

Pappas, et al., “Transparent ROP Exploit Mitigation Using Indirect Branch Tracing” USENIX 2013 (“Pappas”, EX1005) is prior art under 35 USC §

102(a)(1) because it was published at the latest on November 16, 2013 on the internet, before the earliest possible priority date of the '591 patent. *See* EX1017.

Pappas was not submitted to or considered by the Examiner who allowed the '591 patent.

B. Statutory Grounds for Challenge

Petitioner requests review of claims 1-18 on the following grounds:

GROUND 1: Claims 1-18 are unpatentable under 35 U.S.C. § 103 as obvious in view of Dalcher and Pappas.

IV. The '591 Patent

A. Claim of Priority of the '591 patent

The '591 patent, entitled “Detecting and preventing execution of software exploits,” issued on May 7, 2019, from U.S. Patent Application No. 14/606,604, which claims priority to U.S. Provisional Patent Application No. 61/931,772, filed

January 27, 2014. EX1001, Front Cover. Thus, the earliest possible priority date for the '591 patent for the purposes of this IPR is January 27, 2014.

B. Specification of the '591 Patent

The '591 patent describes anti-exploit systems and methods related to monitoring executable computer files (e.g., software) accessing memory to detect and prevent execution of functions that could exhibit suspicious behavior. One or more functions that are stored in memory during execution are monitored by loading a component for evaluating the function. When a call from an application programming instance invokes a function, the methods include executing stack walk processing in the monitored memory space, and performing during the stack walk processing, before an address of the originating caller function is reached, a memory check of stack entries to detect suspicious behavior. An alert of suspicious behavior is issued when the memory check detects at least one of a number of indicators of suspicious behavior. Issuing the alert prevents execution of a payload for the invoked function from operating. EX1001, Claim 1, 13:29-61. The '591 patent includes claims to computer-implemented methods (claim 1-8), claims to systems implementing the methods (claims 9-16), and claims to a computer-readable storage device containing instructions implementing the methods (claims 17-18)

More particularly, claims 1-8 of the '591 patent are directed to computer-implemented methods that include: “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.” EX1001, 13:29-61.

Dependent claim 2 recites a further limitation of identifying an address associated with the invoked function and preventing execution of the invoked function when the address is found in a cache. *Id.*, 13:61-67. Dependent claim 3 recites further limitations regarding how a stack walk is performed. *Id.*, 14:1-5. Dependent claims 4 and 5 recite further limitations regarding how monitoring functions are performed. *Id.*, 14:6-17. Dependent claim 6 lists six types of functions for which the plurality of functions are monitored. *Id.*, 14:18-22. Dependent claim 7 recites further limitations regarding the memory check that is performed in dependent claim 5. *Id.*, 14:23-25. Dependent claim 8 recites preventing execution of a payload of the invoked function when an address of the invoked function is found in a cache. *Id.*, 14:26-28.

Claims 9-16 recite a system comprising at least one memory and at least one processor configured to perform operations that correspond to the methods recited in claims 1-8. EX1001, 14:29-15:33. Claim 17 recites a computer-readable storage device on which are stored instructions that when executed on at least one processor cause the processor(s) to execute processes that generally correspond to the methods recited in claim 1. Claim 18 recites further limitations on the memory checks recited in claim 17. *Id.*, 15:34-16:42.

C. Summary of the Prosecution History

Prosecution of the 14/606,604 application that issued as the '591 patent began with a non-final Office Action on December 16, 2016, continued through two non-final Office Actions, one final Office Action, three sets of claim amendments, and one Request for Continued Examination, ending in a Notice of Allowance that issued on December 20, 2018. EX1006, pp.18–145. The final round of rejections and amendments is illustrative of the Examiner's findings during prosecution.

In a non-final Office Action that issued on August 30, 2018, claims 1-20 were rejected. Specifically, claims 1-6, 8-15 and 17-20 were rejected under 35 U.S.C. § 103(a) as unpatentable over U.S. Patent Publication 2012/0255018 to Salaam in view of U.S. Patent Publication 2007/0016914 to Yeap. EX1006, pp.62–74.

In an amendment filed October 1, 2018, Applicants responded to the non-final Office Action by amending the independent claims. Specifically, the independent claims were amended to recite “...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....”

(underlines showing the amended language) EX1006, pp.40–47. In associated remarks, Applicants argued that Salaam and Yeap do not disclose the limitations added in the claim amendments. *Id.*, pp.48–53. In the Notice of Allowance, the Examiner did not provide an examiner’s statement of reasons for allowance and instead merely noted that the Applicant’s amendments and remarks were persuasive. *Id.*, pp.19–20.

D. Level of Ordinary Skill in the Art

As established by the Declaration of Dr. Seth Nielson dated March 7, 2023, a person having ordinary skill in the art (“PHOSITA”) at the time of the earliest claimed priority date for the ’591 patent would have had 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 topics such as malware/exploit detection. More education can supplement practical experience and vice versa. I EX1002, ¶150.

E. Claim Construction

In this proceeding, claims are interpreted under the same standard applied by Article III courts (i.e., the *Phillips* standard). 37 C.F.R. §42.100(b); *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005) (en banc).

In parallel litigation, Webroot argues all claim terms should be given their plain and ordinary meanings. For purposes of this proceeding only, Petitioner

proposes the Board adopt Webroot's proposal. *See Western Digital Corporation v. Spex Technologies, Inc.*, IPR2018-00084, Paper 14, *11 (PTAB Apr. 25, 2018) (finding Petition based on claim constructions urged by Patent Owner satisfies the claim construction requirements and Petitioner is not required to express its subjective agreement regarding correctness of the proffered claim construction or take ownership of the construction); *see also General Electric v. Vestas*, IPR2018-00928, Paper 9, *16-17 (PTAB Nov. 5, 2018).¹

V. Grounds of Rejection

The following discussion and the accompanying evidence demonstrate that there is a reasonable likelihood that Petitioner will prevail with respect to the challenged claims.

A. Ground 1: Claims 1-18 are unpatentable under 35 U.S.C. § 103(a) as obvious based on Dalcher in view of Pappas.

Petitioner evaluates the scope and content of the prior art, any differences between the art and the claim, and the knowledge of a person of ordinary skill in the art in accordance with *Graham v. John Deere Co.*, 383 U.S. 1 (1966) and *KSR Int'l Co. v. Teleflex, Inc.*, 550 U.S. 398 (2007). Petitioner is unaware of any

¹ Petitioner reserves the right in other proceedings to construe the Challenged Claims and pursue indefiniteness theories.

secondary considerations relevant to the '591 patent and a more than adequate written rationale is provided to support the combination of Dalcher and Pappas under *KSR*. 550 U.S. at 418.

Dalcher is directed to ensuring that “the execution flow of the function may be monitored for malware.” EX1004, ¶¶[0025]; EX1002, ¶¶[153,179]. This monitoring is disclosed as being accomplished via monitoring points (e.g., hooks, call backs, filter drivers) inserted into the call stack of a series of function calls or call frames in memory. EX1004, ¶¶[0025]; EX1002, ¶¶[100-102,159,181]. Dalcher discloses that once a monitoring point is reached or triggered, the monitoring method will walk back through the stack to validate the chain of function calls to identify malicious insertions. EX1004, ¶¶[0037]-[0039]; EX1002, ¶¶[156-158,186]. This process of walking backwards in a memory stack can proceed through the chain of calls using the pointers in the call frames themselves or, if the pointers are not available, the stack walk may proceed backward through sequential memory addresses of the stack towards the bottom of the stack. EX1002, ¶¶[81-85,186-187]; EX1004, ¶¶[0036]-[0039]. A malicious function call, for example, can be identified if it calls out to virtual memory or freed memory where benign functions would likely not call or can be identified by return calls to improper addresses. EX1002, ¶¶[64-84]; EX1004, ¶¶[0036]-[0039]. Thus, Dalcher uses the well-known process of stack walking to verify the chain of function calls to prevent malware

from executing. EX1002, ¶¶79-84,156-168; EX1004, ¶¶[0036]-[0039].

Pappas is directed to identification of malicious attacks inserted or initiated in the memory/call stack of a computer. EX1005, pp.2; EX1002, ¶¶172-174. Specifically, the main threat that Pappas addresses is exploits of return calls in the call stack to execute arbitrary code before returning as originally intended by the legitimate program. EX1005, pp.2-3,7; EX1002, ¶¶169-171. To protect the call stack, Pappas teaches an approach for monitoring branching, control flow transfers, and other key functions in the call stack using a hook. EX1005, pp.3-5,454; EX1002, ¶¶174-176. This implementation (called kBouncer) disclosed in Pappas is based on last branch recording (LBR) which creates a parallel cached stack of the legitimate application calls for comparison in order to identify malicious redirects and insertions. EX1005, pp.4-6,9-10; EX1002, ¶¶186-189. The kBouncer monitor flags illegal calls and returns in order to prevent their execution and then analyzes the gadget/call chain for malicious calls. EX1005, pp.7-10; ¶¶176-177,182-185. Thus, Pappas, like Dalcher, flags and hooks certain monitored functions and then analyzes the chain of calls to identify any abnormalities. EX1005, pp.6-9; EX1002, ¶176.

A PHOSITA would have been motivated to combine the teachings of Dalcher and Pappas since both references relate to detecting malware using the same or similar processes of call stack analysis and return-oriented programming. EX1002,

¶¶179,183-184; EX1004, ¶¶[0020]-[0025]; EX1005, pp.4. Indeed as described above, Dalcher and Pappas both provide stack analysis that targets improper calls in memory and traces call chains in a memory stack to verify proper call initiations. EX1005, pp.6-7; EX1004, ¶¶[0024],[0039]; EX1002, ¶¶181,184,186. A PHOSITA would have implemented Pappas and Dalcher together to address the common exploitation technique of return oriented programming by analyzing the relationships between stack frames (calls). EX1002, ¶¶184-186; EX1005, pp.2-3,8-9; EX1004, ¶¶[0036]-[0037],[0041],[0046]; EX1005, pp.5-7. A PHOSITA would have looked to the teachings of Pappas to modify the teachings in Dalcher in order to enhance the tracking of calls, jumps, and returns and to reduce false positives in compromise detection rates. EX1002, ¶¶185-189; EX1004, ¶¶[0039]-[0043]; EX1005, pp.8-10.

Further, Dalcher and Pappas provide solutions to the same problem – detection of compromises and non-benign conditions in computers – and do so in analogous ways such that a PHOSITA would have confidence that the combination would provide predictably successful results. EX1002, ¶¶106-109. Dalcher teaches analysis of calls preserved in portions of memory called freed memory and teaches further preservation of the call stack for analysis, and, likewise, Pappas teaches a compatible cache (preserved LBR stack) for analysis. EX1002, ¶¶183-189; EX1004, ¶¶[0018],[0036],[0037],[0046]; EX1005, pp.5-7, Fig.-6. Therefore, the

combination of Dalcher and Pappas described in more detail below would have been compatible and recognized as likely to succeed to a PHOSITA.

1. Claim 1

a. A computer-implemented method comprising:

The memory monitoring process of Dalcher operates on a computer system as pictured in FIG. 1 and FIG. 2 of Dalcher. EX1004, ¶¶[0012]-[0016]; EX1002, ¶¶194-198.

Therefore, Dalcher discloses the preamble. *Id.*

b. 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;

Dalcher discloses monitoring a call stack (corresponding to “memory space of a process for execution of at least one monitored function...”) containing one or more functions to be executed (corresponding to “a process”). EX1004, ¶¶[0017]-[0023]; EX1002, ¶¶203-204. A PHOSITA would have understood that the call stack of Dalcher represents a portion of memory including freed memory. EX1004, ¶¶[0023],[0033]; EX1002, ¶¶81-85,204. Dalcher discloses monitoring operating system operations and functions (“monitored function of a plurality of functions”) and mechanisms for monitoring functions (e.g., hooked functions become monitored functions). EX1002, ¶¶205-212; *see also* discussion regarding limitation 1[e]. Further, a PHOSITA would understand that any number of functions may be

called and appear in the memory space or stack to be monitored (corresponding to “monitored function of a plurality of functions”), and that not all of these functions are equally important for monitoring. EX1002, ¶¶111-120,208,; EX1004, [0039][0041]; EX1005, pp.2. Thus, Dalcher explicitly teaches that the monitored function is in the monitored call stack (corresponding to “at least one monitored function in the memory space”). EX1002, ¶¶205-210. To the extent that the claim limitation requires the loaded component (“loading a component”) to also be in the memory space, Dalcher discloses this stating that monitoring points (e.g., hooks) are placed in the call stack being monitored. EX1004, ¶[0033]; EX1002, ¶¶210-212.

Dalcher monitors the execution flow of the call stack using a variety of mechanisms including hooks, filter drivers, event callbacks, and virtualization extensions (corresponding to “a component for evaluating the at least one monitored function in the memory space”). EX1004, ¶¶[0023],[0030]-[0033]; EX1002, ¶¶205-209,211-212. These mechanisms are associated with particular functions such that a hook is inserted by the system of Dalcher for a particular function of a process to enable monitoring of that function. EX1004, ¶¶[0023],[0030]-[0033]; EX1002, ¶¶205-207. For example, filter drivers and code hooks are used to modify the control flow of software for evaluating or monitoring a function’s behavior, by temporarily redirecting execution of a function to a

software component provided by the system that installed the filter driver or code hook. **EX1002, ¶¶95-102 (citing EX1013,EX1011).** Dalcher discloses “the execution flow of the function may include monitoring the call stack frame from the freed memory in the call stack. For example, a monitoring point (e.g., a hook, etc.) may be placed on the call stack frame...” EX1004, ¶¶[0023]; EX1002, ¶¶208-209. A POSITA would have understood based on this disclosure that Dalcher discloses “monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space” (e.g., hook). **EX1002, ¶¶95-102,209.**

Pappas also discloses this limitation, including “wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space.” EX1002, ¶¶214-221. To perform the monitoring, Pappas explains that “kBouncer” module would be loaded ahead of time for monitoring and evaluating protected functions (loading “a component” that monitors and evaluates the function within the call stack memory space). EX1002, ¶¶218-219. Pappas explains that the kBouncer module “hooks the set of protected Windows API functions” (monitored functions), “detours the execution” of such functions, and “inspects the LBR stack” (call stack (memory space) relating to execution of monitored Windows API calls/functions) in order to monitor/evaluate such monitored functions. EX1002, 218-219; EX1005, pp.10..

Thus, Pappas also discloses this limitation. EX1005,p.10,Fig.8; EX1002, ¶¶214-221.

Therefore, Dalcher and Pappas each disclose this limitation. EX1002, ¶¶202-223.

c. invoking one of the plurality of functions as a result of receiving a call from an application programming instance;

A PHOSITA would have understood that the “application programming instance” phrase of this limitation relates to the standard technical term of an application programming interface (API) based on the disclosure in the ’591 specification and knowledge of PHOSITA. EX1002, ¶¶89-90,143-144; EX1001, 3:40-50.

Dalcher teaches that certain events are expected to be handled by OS-supplied functions. EX1002, ¶¶227-228; EX1004, ¶[0026]. Dalcher teaches monitoring these OS-supplied functions to ensure proper handling. *Id.* That is, a call to an OS-supplied (API) function would be understood by a PHOSITA to then invoke that function or another API function, including monitored functions (corresponding to “invoking one of the plurality of functions as a result of receiving a call from an application programming instance”). EX1002, ¶¶136,137,139; EX1004, ¶¶[0026]-[0032]. Further, Dalcher discloses that “code that is part of an interface may be identified through stack analysis as having been invoked,” where such an interface would be understood by a PHOSITA to invoke

functions in response to the API call itself. EX1002, ¶¶229-230; EX1004, ¶[0045]. A PHOSITA would have understood either of these references to be a disclosure of an operating system API invoking functions (including monitored functions) and doing so in response to calls to its functions (corresponding to “invoking...as a result of receiving a call from an application programming instance”). EX1002, ¶¶89-90,226-230; EX1004, ¶¶[0026]-[0032].

To the extent that a call from an API is not explicitly disclosed in Dalcher, Pappas teaches this limitation. Pappas teaches monitoring system calls (syscalls) of the Windows operating system (OS) that are invoked by calls to the Windows API. EX1002, ¶¶232-233; EX1005, pp.5-6. Indeed, the primary way that syscalls are initiated is through a call to the Windows API which operates as a form of protection for these syscalls. EX1002, ¶¶86-90(citing EX1008,EX1014,EX1016); EX1005, pp.5-6. Thus, the kBouncer tool of Pappas teaches this invocation process which was routinely performed by the Windows OS at the time. EX1002, ¶¶234-238; EX1005, pp.10,Fig.-8.

Therefore, Dalcher in view of Pappas discloses this limitation. EX1002, ¶¶225-238.

d. executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and

For purposes of this petition, Petitioner is adopting the Patent Owner’s plain and ordinary meaning for “stack walk processing” that was alleged in District

Court litigation, which is that stack walk processing is processing that involves using a stack walk.

A “stack walk processing” may include setup and configuration for an actual stack walk that begins later. EX1002, ¶242. Dalcher teaches a stack walk as stack analysis as follows:

performing the stack analysis may include identifying one or more call frames of the call stack by performing a stack walk. For example, stack frames may be discovered by walking back in the stack when stack frame pointers are not present.

EX1004, ¶[0035].

Dalcher further discloses that this stack analysis or stack monitoring including stack walk (corresponding to “stack walk processing”) is applicable to or may be carried out as a result of a monitored (e.g., hooked) function been invoked (“invocation of one of the plurality of functions”), which triggers other processes (e.g., an additional hook/monitoring point being placed on a subsequent function (e.g., function that may be called later) and when that subsequent function is called, performing stack walk from that function to the initially invoked function to analyze stack frames of a stack) (referred to as stack analysis and extracted stack analysis by Dalcher). EX1004, ¶[0034]; EX1002, ¶¶248-253, 260-264. That is, the stack walk disclosed and described with reference to FIG. 5 of Dalcher is a form of

monitoring that is called or triggered by another function being called or a frame being added to the stack (e.g., via hook, callback, or filter driver). EX1002, ¶251.

Specifically, Dalcher discloses that further analysis may be performed if “monitoring indicates that that other software was involved in this handling.” EX1004, ¶[0026]; EX1002, ¶¶252-253. That is, Dalcher teaches performing more stack analysis if other software is invoked in the monitored test event in the stack (corresponding to “invocation of one of the plurality of functions”), and further discloses that this further/extended analysis includes a stack walk. EX1004, ¶¶[0026]-[0034]; EX1002, ¶¶250-253. Likewise, Dalcher teaches “if ...it is determined that the callback is within the scope of the monitoring, then in operation 410, a call stack for the operation that triggered the monitoring callback is extracted and examined”(corresponding to “invocation of one of the plurality of functions”). EX1004, ¶¶ [0030]-[0034]; EX1002, ¶257. A PHOSITA would have understood that the monitoring call back for further analysis taught by Dalcher includes monitoring via stack walk upon invocation of a function as required by this limitation . EX1002, ¶¶244-247,257,261-264.

To the extent that Dalcher is considered to not execute stack walk processing upon invocation of a function, Pappas also teaches limitation 1[d]. For example, Pappas discloses Windows API function as an originally monitored function. **EX1002, ¶¶219-221,233-237; EX1005, pp.5,10,Fig.-8.**When the monitored API

function is invoked, Pappas executes stack walk processing by adding monitoring for a syscall (function) that is meant to be triggered/called by the API so that stack walk can proceed from the syscall to the invoked function (API function). EX1002, ¶¶254-265.

Therefore, Dalcher discloses this limitation. EX1002, ¶241-266.

- e. 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;**

Dalcher discloses “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.”

Dalcher discloses executing a stack walk that walks back in the stack from a function call (e.g., a current top of the stack). EX1004, ¶¶[0036]-[0037]; EX1002, ¶¶270-275. A PHOSITA would have understood that the top of a stack corresponds to the most recently called function. EX1002, ¶¶62-85(citing EX1008, EX1012, EX1009). The stack walk disclosed in Dalcher traces a chain of function calls (stack entries of the stack) from most recently called function

(subsequently-called function) back through stack to an originating function / initially invoked function (“originating caller function”). EX1004, ¶¶[0036]-[0037]; EX1002, ¶271. This tracing direction from recent calls to less recent calls and on to an initially called function is an essential aspect of walking back in a call stack, which is in a direction opposite to the call order (e.g., from subsequently-called function back to originally monitored/hooked (and called) function). EX1002, ¶¶62-85,91-95,270-276(citing EX1015,EX1008).

Further, Dalcher discloses that a monitored function being invoked can trigger additional monitoring including checks or analysis of memory (corresponding to “memory check”) in the stack. EX1002, ¶274; EX1004, ¶[0033]. That is, the invocation of one function can trigger additional monitoring or a stack walk (including memory checks) from a memory address (e.g., of a subsequently-called function) in the stack that has been saved for monitoring (e.g., by a hook, probe).

A PHOSITA would have recognized that the additional monitoring (including stack walk and memory checks/analysis) and hooking after initial invocation and monitoring teaches the features of “performing... a memory check for a plurality of stack entries identified during the stack walk processing.” EX1002, ¶¶62-85,91-95,270-274; EX1004, ¶¶[0030]-[0034]. Further, a PHOSITA would have understood that because a stack walk walks back from a subsequently-

called function to the initially invoked function (originating caller function), memory checks of stack entries performed during the stack walk occur before the address of the originating caller function is reached. EX1002, ¶¶62-85,91-95,255-257,270-274; EX1004, ¶¶[0030]-[0034].

Dalcher similarly discloses that the contents of frames of the stack can be analyzed as a part of the stack walk to determine if contents have been overwritten, which is an indicator of compromise/suspicious behavior (corresponding to “memory check for...stack entries identified during the stack walk processing to detect suspicious behavior”). EX1004, ¶[0024]; EX1002, ¶¶270-271,277,281,286. The monitoring disclosed by Dalcher includes determining if a compromise occurred and a level of confidence in that determination, as well as generating a flag indicating detection of such compromise (corresponding to “alert of suspicious behavior is triggered”). EX1004, ¶¶[0024]-[0026]; EX1002, ¶¶287-289.

Second, Dalcher also discloses detection (e.g., through memory checks performed during a stack walk) of at least two of the elements of the list of alternatives that follows the colon of this limitation (hereinafter referred to as “alternative elements”). Dalcher discloses evaluating frames of a stack to determine if they have been overwritten (corresponding to “a base pointer is identified as being invalid” as well as “an invalid stack return address is identified”). EX1004, ¶[0024]; EX1002, ¶¶289-290,292. Dalcher teaches analyzing

code preceding a return pointer in a frame which may indicate that the return pointer is invalid or overwritten (corresponding to “an invalid stack return address is identified”). *Id.* Accordingly, Dalcher teaches at least two of the alternative elements of this limitation.

Pappas teaches monitoring of a memory stack (including a stack walk and memory checks) that is consistent with Dalcher and also teaches at least one of the alternatives elements of this limitation. EX1005, pp.6; EX1002, ¶¶161-162,165,172. Specifically, Pappas teaches performing a stack walk and other analysis to identify “abnormal control transfers that take place during [return-oriented programming] ROP code execution” (corresponding to “attempted execution of a return-oriented programming technique is detected”). EX1005, pp.3; EX1002, ¶¶293-297.

Moreover, Dalcher and Pappas disclose “wherein an alert of suspicious behavior is triggered when the...memory check detects” the above-discussed alternative elements. For example, Dalcher discloses putting a system in quarantine when the above-discussed alternative elements are detected, which a PHOSITA would have understood to involve triggering an alert of suspicious behavior. Also, Pappas displays a visual alert of suspicious behavior when the above-discussed alternative element is detected. EX1002, ¶¶288-292, 296-297.

Therefore, Dalcher discloses elements listed in this limitation and Pappas discloses compatible and consistent memory stack monitoring, any of which are sufficient to render the limitation obvious. EX1002, ¶¶268-299,292,297,299; *See e.g., Titanium Metals Corp. of Am. v. Banner*, 778 F.2d 775,782 (Fed. Cir. 1985)(“It is also an elementary principle of patent law that when...a claim covers several compositions, the claim is "anticipated" if one of them is in the prior art.”) .

f. wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function from operating.

Dalcher discloses monitoring to determine whether a compromise (suspicious behavior) occurred and a level of confidence in the determination, and if the level of confidence that a compromise has occurred is above a certain threshold, a flag is generated indicating the compromise (corresponding to “alert of suspicious behavior is triggered”). EX1004, ¶¶[0024]-[0026]; EX1002, ¶¶303-304. Dalcher discloses placing software or functions in quarantine that have been flagged as compromised (corresponding to “preventing ...from operating”). EX1004, ¶[0026]; EX1002, ¶304.

A PHOSITA would have understood that the quarantine process of Dalcher involves preventing operation and ultimately execution of quarantined software or functions (including payload for the invoked function). EX1002, ¶¶304. Pappas discloses terminating execution of code or frames (corresponding to “preventing execution of a payload for the invoked function”) that have been identified by

memory analysis to be compromised/suspicious. EX1005, pp.455; EX1002, ¶¶305-306. A PHOSITA would have looked to Pappas for further compatible details on preventing execution of quarantined software or functions when a compromise (suspicious behavior) is detected. EX1002, ¶¶305-307.

Therefore, Dalcher and Pappas disclose this limitation. EX1002, ¶¶302-308.

For the foregoing reasons, a PHOSITA as of the earliest possible priority date would have found claim 1 of the '591 patent to be obvious in view of the combination of Dalcher and Pappas because these prior art publications disclose each and every limitation recited in the claim. *Id.* Accordingly, claim 1 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher and Pappas.

2. Claim 2

The method according to claim 1, further comprising:

a. identifying an address associated with the invoked function.

Dalcher teaches monitoring a memory stack including return addresses. EX1004, ¶¶[0018],[0024]; EX1002, ¶¶313-314. Dalcher also discloses placing monitoring hooks and probes into the call stack to identify addressing and function flow, where only invoked functions are on a call stack. EX1004, ¶¶[0019],[0020],[0023]; EX1002, ¶¶314-315. A PHOSITA would have understood that these monitoring techniques identify an address of a function call. EX1002, ¶¶315. Pappas discloses consistent and compatible address identification and a PHOSITA

would have looked to the more detailed implementation of Pappas to reduce false positives. EX1002, ¶¶316-321; EX1005, pp.2-6.

Therefore, Dalcher in view of Pappas discloses this limitation. EX1002, ¶322.

b. when the address of the invoked function address is found in a cache, preventing execution of a payload for the invoked function.

Pappas teaches analyzing call stacks and caching one or more function calls via a process called last branch recording (LBR). EX1005, pp.3-6; EX1002, ¶329. Pappas records a parallel LBR stack containing indirect memory branches together with addresses in an LBR cache. *Id.* Pappas then analyzes these stored branches (in modified form) by searching for an address corresponding to the target address about to be executed (corresponding to “the address of the invoked function address is found in a cache”). EX1005, pp.9-11; EX1002, ¶¶329-333.

Further, a PHOSITA would have understood that the “freed memory” analyzed in Dalcher is a cache of recently executed functions, since this freed memory preserved function calls for analysis. EX1002, ¶¶325-328; EX1004, ¶[0021],[0037]-[0038]; EX1007, pp.15-16. Likewise, a PHOSITA would have understood that such executed functions (e.g., including the invoked function), the present stack, and branches of functions can be cached in separate memory as disclosed in Pappas. EX1005, pp.3-6; EX1002, ¶¶329-333. Dalcher also teaches the need to preserve executed functions for analysis, which would motivate a

PHOSITA to look to Pappas for further details regarding how to perform such preservation. EX1004, ¶¶0046]; EX1002, ¶¶333-337. Accordingly, a PHOSITA would have understood that the cache disclosed in Dalcher and the cache disclosed in Pappas are complementary and compatible memory monitoring processes. EX1002, ¶¶336-340.

Further, as described regarding limitation 1[f], Pappas discloses terminating execution of code or frames (corresponding to “preventing execution of a payload for the invoked function”) identified by memory analysis to be compromised. EX1005, pp.455; EX1002, ¶¶293-297. Again, a PHOSITA would have understood that the quarantine process of Dalcher involves preventing operation and ultimately execution of compromised software or functions. EX1002, ¶¶288-292,340. A PHOSITA would have looked to Pappas for further compatible details on preventing execution when a condition indicative of compromise (e.g., address of an invoked function address is found in a cache) is detected. EX1002, ¶340. A PHOSITA would have been motivated to use the techniques disclosed in Pappas for reducing detection false positives. EX1002, ¶¶297,340; EX1005, pp.7-10.

For these reasons, the combination of Dalcher and Pappas discloses this limitation. EX1002, ¶341.

Therefore, a PHOSITA would have found the limitations recited in claim 2 obvious in view of the Dalcher-Pappas combination. EX1002, ¶310-341. As

shown, the Dalcher-Pappas combination discloses all elements of claim 1 from which claim 2 depends. Accordingly, claim 2 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

3. Claim 3

The method according to claim 1, wherein executing of the stack walk processing further comprises walking a stack frame associated with the invoked function from a low level user mode function to the originating address of the invoked function.

As noted above for Claim 1, Dalcher discloses a stack walk as part of its memory analysis that is performed by walking back through the list of calls and frames that form the stack (corresponding to “executing of the stack walk processing further comprises walking a stack frame associated with the invoked function”). EX1004, ¶¶[0034]-[0035]; EX1002, ¶¶62-85,91-94,242-248,343. A PHOSITA would have understood that “walking a stack frame associated with the invoked function” involves following the chain of calls or branches associated with the stack frame/stack. EX1002, ¶343.

Further, Pappas teaches a form of stack walking that includes progressing through the chain of calls from lower-level calls to higher-level calls. EX1002, ¶¶344-345; EX1005, pp.10. Pappas discloses monitoring calls to the Windows API (corresponding to “low level user mode function”), which is a user-level function that is less abstracted than application-level calls. EX1002, ¶¶344-345; EX1005, pp.5-6,9-10. Pappas states “system calls are always invoked solely through their

respective Windows API” which establishes a hierarchy of application-level functions (corresponding to “originating address of the invoked function”), down to the Windows API functions, down to various system-level calls. *Id.* A PHOSITA would have understood that Dalcher and Pappas’ stack walk can follow the chain of calls from a Windows API call to an originating application (originating address of the invoked function) in the user level. *Id.*

A PHOSITA would have understood how to apply the call-chain analysis of Pappas to the memory stack of Dalcher at least because both are based on chains of calls and both contemplate walking back in the stack of memory calls; namely, Pappas discloses hooking and inspecting the stack, and Dalcher teaches hooking and walking back in the stack. EX1002, ¶¶343-347; EX1005, pp.10; EX1004, ¶¶[0036],[0046]. A PHOSITA would have been motivated to apply the stack analysis across abstraction levels as disclosed in Pappas with the stack walk of Dalcher in order to obtain better insight into return-oriented programming calls and API calls. EX1002, ¶¶343-347; EX1005, pp.9-11.

Therefore, a PHOSITA would have found the limitations recited in claim 3 obvious in view of the Dalcher-Pappas combination. EX1002, ¶342-351. As shown, the Dalcher-Pappas combination discloses all elements of claim 1 from which claim 3 depends. Accordingly, claim 3 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

4. Claim 4

The method according to claim 1, wherein the monitoring further comprises monitoring functions associated with memory management in the memory space by a user mode function.

Further to the memory monitoring of Dalcher described above regarding claim elements 1[b], 1[c], and 1[d], Dalcher discloses initiating test events in a specific thread (corresponding to “monitoring ...by a user mode function”) and monitoring handling of that event by the memory. EX1004, ¶¶[0027]-[0029]; EX1002, ¶¶353,355-356.

Likewise, Pappas teaches monitoring a large set of memory functions at the OS-level (e.g., Windows APIs) including monitoring memory functions associated with memory management (e.g., VirtualAlloc, CreateProcessA, etc.). EX1002, ¶354; EX1005, pp.13. A PHOSITA would have understood the monitored memory functions listed in Pappas to relate to memory management in the memory space by a user mode function. EX1002, ¶¶357-359. A PHOSITA would have been motivated to modify the teachings of Dalcher to implement the monitoring as described for the memory management functions in Pappas at least because Pappas discloses that such functions are commonly targeted for compromise. EX1002, ¶354; EX1005, pp.13. A PHOSITA would have been able to combine the OS-level memory monitoring of Dalcher with the memory management targeting of Pappas at least because Dalcher supports compatible targeting and Pappas provides

specific functions to target with such monitoring. EX1002, ¶¶358-359; EX1005, pp.458; EX1004, ¶¶[0025]-[0026].

Therefore, a PHOSITA would have found the limitations recited in claim 4 obvious in view of the Dalcher-Pappas combination. EX1002, ¶¶353-360. As shown, the Dalcher-Pappas combination discloses all elements of claim 1 from which claim 4 depends. Accordingly, claim 4 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

5. Claim 5

The method according to claim 1, wherein the monitoring further comprises applying, upon detecting a function call of one of the plurality of functions in the monitored memory space, a hooking technique to hook a lowest level user mode function of the invoked function, wherein the executing of the stack walk processing executes a reverse stack walk from the lowest level user mode function of the invoked function.

As noted regarding claim limitation 1[b], Dalcher teaches applying hooks (corresponding to “hooking technique”) to trigger processes when a function is called. EX1004, ¶¶[0020],[0023]-[0024],[0032],[0033]; EX1002, ¶¶[362-363; *see also* discussions regarding limitations 1[c]-1[e] above. Dalcher also teaches applying further memory analysis techniques (e.g., including applying a hook to a function that may be called after a monitored function is invoked) after detecting suspicious activity via initial monitoring / when a monitored function is initially invoked (corresponding to “upon detecting a function call of one of the plurality of functions in the monitored memory space, a hooking technique to hook...”). *Id.*

Specifically, Dalcher teaches performing test events and then performing additional monitoring of operating system handling. EX1004, ¶¶[0026]-[0033],[0036]-[0037]; EX1002, ¶¶363-364. Dalcher also teaches performing a stack walk from the most recently called function (top of stack) back to the prior called functions. *Id.* A PHOSITA would have understood that in the context of a chain of function calls in a stack, a stack walk from a hooked function or top of the stack would be a reverse stack walk back to a function that called the hooked function. EX1002, ¶¶362-364; EX1004, ¶¶[0035]-[0037]; *see also* discussions regarding limitations 1[b]-1[e] above.

Pappas teaches a user-level function that calls an Operating System (OS)-level API that calls a system-level function (corresponding to “lowest level user-mode function”). EX1005, pp.10; EX1002, ¶¶365-366. Pappas discloses tracing the origin of hooked system calls to Windows API calls and using hooks and checkpoints to monitor invocation of system calls (corresponding to “applying...hooking techniques to hook a lowest level user-mode function”). EX1005, pp.5-6; EX1002, ¶¶365-367. A PHOSITA would have understood that the teachings of Dalcher and Pappas involve reverse stack walks that walk back in reverse order of the stack. EX1002, ¶¶367-372. A PHOSITA would have been motivated to implement the system call and API hooks disclosed in Pappas to build on the monitoring disclosed in Dalcher at least because exploitation of such

processes is common and the Pappas methods reduce false positives in detection.

EX1002, ¶¶371-374; EX1005, pp.2,6.

Therefore, a PHOSITA would have found the limitations recited in claim 5 obvious in view of the Dalcher-Pappas combination. EX1002, ¶¶362-375. As shown, the Dalcher-Pappas combination discloses all elements of claim 1 from which claim 5 depends. Accordingly, claim 5 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

6. Claim 6

The method according to claim 1, wherein the plurality of functions comprises an NtAllocateVirtualMemory function, an NtProtectVirtualMemory function, an NtCreateProcess function, an NtCreateProcessEX function, an NtCreateUserIProcess function, and an LdrLoadDll function.

As described regarding claim limitation 1[c], Dalcher discloses inserting monitoring points into the stack to monitor invocation of certain functions. EX1004, ¶¶[0019]-[0020],[0023],[0025]; EX1002, ¶¶377. Pappas discloses over 52 different memory functions that should be monitored, including CreateProcessA, CreateProcessW, VirtualAlloc, VirtualProtect, LoadLibraryA, LoadLibraryW. EX1005, pp.13,17; EX1002, ¶¶378-380. A PHOSITA would have understood that these monitored functions correspond to “NtCreateProcess, NtCreateUserIProcess, NtCreateProcessEX, NtAllocateVirtualMemory, NtProtectVirtualMemory, and LdrLoadDll” recited in claim 6, respectively. EX1002, ¶¶378-380; EX1005, pp.10,13. A PHOSITA would have understood that

to the extent any of the recited functions deviate from those disclosed by Pappas, that protection of such known functions is obvious based on their operational similarity. EX1002, ¶¶380-384. Further, the motivation disclosed in Pappas for protecting the 52 list functions would be understood by a PHOSITA to extend to the recited functions. *Id.*

Therefore, a PHOSITA would have found the limitations recited in claim 6 obvious in view of the Dalcher-Pappas combination. EX1002, ¶¶377-385. As shown, the Dalcher-Pappas combination discloses all elements of claim 1 from which claim 6 depends. Accordingly, claim 6 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

7. Claim 7

The method according to claim 5, wherein performing the memory check comprises evaluating state information associated with a cache entry for the hooked function.

As discussed regarding Claim 1, Dalcher discloses monitoring memory and a call stack via monitoring points, probes, test events, and hooks in executable code and performing memory checks during a stack walk (corresponding to “memory check”). EX1004, ¶¶[0019]-[0024]; EX1002, ¶¶387-390. Dalcher discloses analyzing (e.g., through memory checks performed during a stack walk) a memory stack including freed memory of the stack, which a PHOSITA would understand to be a cache of recently executed functions. EX1004, ¶¶[0023],[0025]-[0026],[0037]-[0039]; EX1002, ¶¶391-392. Dalcher also discloses preserving

snapshots of functions for analysis of the states of such functions (corresponding to “evaluating state information”). EX1004, ¶¶[0018],[0046]; EX1002, ¶¶391-392. A PHOSITA would have understood that the preserved snapshot of a function that has been hooked as disclosed in Dalcher is a cache entry of the hooked function. EX1002, ¶¶391-399; EX1007, pp.15-16.

Therefore, a PHOSITA would have found the limitations recited in claim 7 obvious in view of the Dalcher-Pappas combination. EX1002, ¶¶387-400. As shown, the Dalcher-Pappas combination discloses all elements of claims 1 and 5, from which claim 7 depends. Accordingly, claim 7 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

8. Claim 8

The method according to claim 7, wherein when an address of the invoked function is found in a cache, preventing execution of a payload for the invoked function.

As discussed regarding limitation 2[b], Pappas teaches analyzing call stacks and caching one or more function calls via a process called last branch recording (LBR). EX1005, pp.3-6; EX1002, ¶¶329-336. Pappas records a parallel LBR stack containing indirect memory branches together with addresses in an LBR cache. *Id.* Pappas analyzes these stored branches (in modified form) by searching for an address corresponding to the target address about to be executed (corresponding to

“the address of the invoked function address is found in a cache”). EX1005, pp.10; EX1002, ¶¶329-336.

A PHOSITA would have understood that the “freed memory” analyzed in Dalcher is a cache of recently executed functions. EX1002, ¶¶325-328; EX1004, ¶¶[0021]-[0023]. Likewise, a PHOSITA would have understood that such executed functions, the present stack, and branches of functions can be cached in separate memory as disclosed in Pappas. EX1005, pp.448-451; EX1002, ¶¶406-411. Dalcher also teaches the need to preserve executed functions for analysis, which would motivate a PHOSITA to look to Pappas for further details on caching (e.g., caching branches). EX1004, ¶¶[0018],[0046]; EX1002, ¶¶406-411.

Further, as described regarding limitation 1[f], Pappas discloses terminating execution of code or frames (corresponding to “preventing execution of a payload for the invoked function”) that have been identified by memory analysis to be compromised. EX1005, pp.10; EX1002, ¶¶408-409. A PHOSITA would have understood that the quarantine process of Dalcher involves preventing operation and ultimately execution of the function. EX1002, ¶¶293-295. A PHOSITA would have looked to Pappas for further compatible details on preventing execution when a compromise is detected. EX1002, ¶¶304-306. A PHOSITA would have been motivated to use the techniques disclosed in Pappas for reducing detection false positives. EX1002, ¶¶78,319-321,408-411; EX1005, pp.8-9.

Therefore, a PHOSITA would have found the limitations recited in claim 8 obvious in view of the Dalcher-Pappas combination. EX1002, ¶¶402-412. As shown, the Dalcher-Pappas combination discloses all elements of claims 1, 5, and 7, from which claim 8 depends. Accordingly, claim 8 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

9. Claim 9

A system comprising:

a. at least one memory; and

Dalcher discloses random access memory (RAM) as part of the computer system that performs the processes of Dalcher. EX1002, ¶¶415-416; EX1004, ¶¶0012].

Therefore, Dalcher discloses this limitation. *Id.*

b. at least one processor connected with the memory configured to perform operation of:

Dalcher discloses a central processing unit (CPU) and microprocessor for executing the processes of Dalcher, where the CPU is connected to RAM in FIG. 2 of the disclosure. EX1002, ¶¶416-417; EX1004, ¶¶0012], Fig.-2. 414. Dalcher discloses that its processes can be implemented as instructions for a device such as a workstation with a processor connected with memory. EX1002, ¶417; EX1004, ¶¶0014],[0015]. A POSITA would have understood that running a program (or processes or instructions) on a computer with a processor involves having the

program instructions executed by the processor to perform the processes (e.g., of Dalcher-Pappas such as those that disclose the processes recited by Claim 9 and its dependent claims), or would have been motivated to configure it in such a manner. EX1002, ¶¶415-417.

Therefore, Dalcher discloses this limitation. *Id.*

c. 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;

Dalcher discloses monitoring a call stack (corresponding to “memory space of a process for execution of at least one monitored function...”) containing one or more functions to be executed (corresponding to “a process”). EX1004, ¶¶[0017]-[0023]; EX1002, ¶¶203-204. A PHOSITA would have understood that the call stack of Dalcher represents a portion of memory including freed memory. EX1004, ¶¶[0023],[0033]; EX1002, ¶¶81-85,204. Dalcher discloses monitoring operating system operations and functions (“monitored function of a plurality of functions”) and mechanisms for monitoring functions (e.g., hooked functions become monitored functions). EX1002, ¶¶205-212; *see also* discussion regarding limitation 1[e]. Further, a PHOSITA would understand that any number of functions may be called and appear in the memory space or stack (corresponding to “monitored function of a plurality of functions”), and that not all of these functions are equally

important for monitoring. EX1002, ¶¶111-120,208,413; EX1004, [0039]-[0041]; EX1005, pp.2. Thus, Dalcher explicitly teaches that the monitored function is in the monitored call stack (corresponding to “at least one monitored function in the memory space”). EX1002, ¶¶205-210. To the extent that the claim limitation requires the loaded component (“loading a component”) to also be in the memory space, Dalcher discloses this stating that monitoring points (e.g., hooks) are placed in the call stack being monitored. EX1004, ¶[0033]; EX1002, ¶¶210-212,421.

Dalcher monitors the execution flow of the call stack using a variety of mechanisms including hooks, filter drivers, event callbacks, and virtualization extensions (corresponding to “a component for evaluating the at least one monitored function in the memory space”). EX1004, ¶¶[0023],[0030]-[0033]; EX1002, ¶¶205-209,211-212,421. These mechanisms are associated with particular functions such that a hook is inserted by the system of Dalcher for a particular function of a process to enable monitoring of that function. EX1004, ¶¶[0023],[0030]-[0033]; EX1002, ¶¶205-207,421. For example, filter drivers and code hooks are used to modify the control flow of software for evaluating or monitoring a function’s behavior, by temporarily redirecting execution of a function to a software component provided by the system that installed the filter driver or code hook. **EX1002**, ¶¶95-102. Dalcher discloses “the execution flow of the function may include monitoring the call stack frame from the freed memory in

the call stack. For example, a monitoring point (e.g., a hook, etc.) may be placed on the call stack frame...” EX1004, ¶¶10023; EX1002, ¶¶208-209. A POSITA would have understood based on this disclosure that Dalcher discloses “monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space” (e.g., hook). EX1002, ¶¶210-221.

Pappas also discloses this limitation, including “wherein monitoring the memory space comprises loading a component for evaluating the at least one monitored function in the memory space.” EX1002, ¶¶214-221. To perform the monitoring, Pappas explains that “kBouncer” module would be loaded ahead of time for monitoring and evaluating protected functions (loading “a component” that monitors and evaluates the function within the call stack memory space). EX1005, pp.5,10. Pappas explains that the kBouncer module “hooks the set of protected Windows API functions” (monitored functions), “detours the execution” of such functions, and “inspects the LBR stack” (call stack (memory space) relating to execution of monitored Windows API calls/functions) in order to monitor/evaluate such monitored functions. EX1005, p.10, Fig.8; Thus, Pappas also discloses this limitation. EX1002, ¶¶214-221.

Therefore, Dalcher and Pappas disclose this limitation. EX1002, ¶¶202,413,421.

d. invoking one of the plurality of functions as a result of receiving a call from an application programming instance;

A PHOSITA would have understood that the “application programming instance” phrase of this limitation relates to the standard technical term of an application programming interface (API) based on the disclosure in the ’591 specification and knowledge of a PHOSITA. EX1002, ¶¶89-90,143-144; *see* EX1001, 3:40-50.

Dalcher teaches that certain events are expected to be handled by OS-supplied functions. EX1002, ¶¶227-228,413; EX1004, ¶[0026]. Dalcher teaches monitoring these OS-supplied functions to ensure proper handling. *Id.* That is, a call to an OS-supplied (API) function would be understood by a PHOSITA to then invoke that function or another API function, including monitored functions (corresponding to “invoking one of the plurality of functions as a result of receiving a call from an application programming instance”). EX1002, ¶¶136,137,139; EX1004, ¶¶[0026]-[0032]. Further, Dalcher discloses that “code that is part of an interface may be identified through stack analysis as having been invoked,” where such an interface would be understood by a PHOSITA to invoke functions in response to the API call itself. EX1002, ¶¶229-230; EX1004, ¶[0045]. A PHOSITA would have understood either of these references to be a disclosure of an operating system API invoking functions (including monitored functions) and doing so in response to calls to its functions (corresponding to “invoking...as a

result of receiving a call from an application programming instance”). EX1002, ¶¶89-90,226-230,413; EX1004, ¶¶[0026]-[0032].

To the extent that a call from an API is not explicitly disclosed in Dalcher, Pappas teaches this limitation. Pappas teaches monitoring system calls (syscalls) of the Windows operating system (OS) that are invoked by calls to the Windows API. EX1002, ¶¶232-233; EX1005, pp.5-6. The primary way that syscalls are initiated is through a call to the Windows API which operates as a form of protection for these syscalls. EX1002, ¶¶86-90; EX1005, pp.5-6. Thus, the kBouncer tool of Pappas teaches this invocation process which was routinely performed by the Windows OS at the time. EX1002, ¶¶234-238,413; EX1005, pp.10, Fig.-8.

Therefore, Dalcher in view of Pappas discloses this limitation. EX1002, ¶¶234-238,413.

e. executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and

For the purposes of this petition, Petitioner is adopting the Patent Owner’s plain and ordinary meaning for “stack walk processing” that was alleged in District Court, which is that stack walk processing is processing that involves using a stack walk.

A “stack walk processing” may include setup and configuration for an actual stack walk that begins later. EX1002, ¶242,413. Dalcher teaches a stack walk as stack analysis as follows:

performing the stack analysis may include identifying one or more call frames of the call stack by performing a stack walk. For example, stack frames may be discovered by walking back in the stack when stack frame pointers are not present. EX1004, ¶[0035].

Dalcher further discloses that this stack analysis or stack monitoring including a stack walk (corresponding to “stack walk processing”) is applicable to or may be carried out as a result a monitored (e.g., hooked) function been invoked (“invocation of one of the plurality of functions”), which triggers other processes (e.g., an additional hook/monitoring point being placed on a subsequent function (e.g., function that may be called later) and when that subsequent function is called, performing a stack walk from that function to the initially invoked function to analyze stack frames of a stack) (referred to as stack analysis and extracted stack analysis by Dalcher). EX1004, ¶[0034]; EX1002, ¶¶248-253,260-264. That is, the stack walk disclosed and described with reference to FIG. 5 of Dalcher is a form of monitoring that is called or triggered by another function being called or a frame being added to the stack (e.g., via hook, callback, or filter driver). EX1002, ¶251,413.

Specifically, Dalcher discloses that further analysis may be performed if “monitoring indicates that that other software was involved in this handling.” EX1004, ¶[0026]; EX1002, ¶¶252-253. That is, Dalcher teaches performing more stack analysis if other software is invoked in the monitored test event in the stack (corresponding to “invocation of one of the plurality of functions”), and further discloses that this further/extended analysis includes a stack walk. EX1004, ¶¶[0026]-[0034]; EX1002, ¶¶250-253,413. Likewise, Dalcher teaches “if ...it is determined that the callback is within the scope of the monitoring, then in operation 410, a call stack for the operation that triggered the monitoring callback is extracted and examined”(corresponding to “invocation of one of the plurality of functions”). EX1004, ¶¶ [0030]-[0034]; EX1002, ¶257. A PHOSITA would have understood that the monitoring call back for further analysis taught by Dalcher includes monitoring via stack walk upon invocation of a function as required by this limitation. EX1002, ¶¶244-247,257,261-264,413.

To the extent that Dalcher is considered to not execute stack walk processing upon invocation of a function, Pappas also teaches limitation 1[d]. For example, Pappas discloses Windows API function as an originally monitored function. EX1005, pp.10. When the monitored API function is invoked, Pappas executes stack walk processing by adding monitoring for a syscall (function) that is meant to

be triggered/called by the API so that stack walk can proceed from the syscall to the invoked function (API function). EX1002, ¶¶254-265,413.

Therefore, Dalcher and Pappas disclose this limitation. EX1002, ¶248,421.

- f. 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,**

Dalcher discloses “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.”

Dalcher discloses executing a stack walk that walks back in the stack from a function call (e.g., a current top of the stack). EX1004, ¶¶[0036]-[0037]; EX1002, ¶¶270-275. A PHOSITA would have understood that the top of a stack corresponds to the most recently called function. EX1002, ¶¶62-85. The stack walk disclosed in Dalcher traces a chain of function calls (stack entries of the stack) from most recently called function (subsequently-called function) back through stack to an originating function / initially invoked function (“originating caller function”).

EX1004, ¶¶[0036]-[0037]; EX1002, ¶271. This tracing direction from recent calls to less recent calls and on to an initially called function is an essential aspect of walking back in a call stack which is in a direction opposite to the call order (e.g., from subsequently-called function back to originally monitored/hooked (and called) function). EX1002, ¶¶62-85,91-95,270-276.

Further, Dalcher discloses that a monitored function being invoked can trigger additional monitoring including checks or analysis of memory (corresponding to “memory check”) in the stack. EX1002, ¶274; EX1004, ¶[0033]. That is, the invocation of one function can trigger additional monitoring or a stack walk (including memory checks) from a memory address (e.g., of a subsequently-called function) in the stack that has been saved for monitoring (e.g., by a hook, probe).

A PHOSITA would have recognized that the additional monitoring (including stack walk and memory checks/analysis) and hooking after initial invocation and monitoring teaches the features of “performing... a memory check for a plurality of stack entries identified during the stack walk processing.” EX1002, ¶¶62-85,91-95,270-274; EX1004, ¶¶[0030]-[0034]. Further, a PHOSITA would have understood that because a stack walk walks back from a subsequently-called function to the initially invoked function (originating caller function), memory checks of stack entries performed during stack walk occur before the

address of the originating caller function is reached. EX1002, ¶¶62-85,91-95,255-257,270-274; EX1004, ¶¶[0030]-[0034].

Dalcher similarly discloses that the contents of frames of the stack can be analyzed as a part of the stack walk to determine if contents have been overwritten, which is an indicator of compromise/suspicious behavior (corresponding to “memory check for...stack entries identified during the stack walk processing to detect suspicious behavior”). EX1004, ¶[0024]; EX1002, ¶¶270-271,277,281,286. The monitoring disclosed by Dalcher includes determining if a compromise occurred and a level of confidence in that determination, as well as generating a flag indicating detection of such compromise (corresponding to “alert of suspicious behavior is triggered”). EX1004, ¶¶[0024]-[0026]; EX1002, ¶¶287-289,413.

Second, Dalcher also discloses detection (e.g., through memory checks performed during the stack walk) of at least two of the elements of the list of alternatives that follows the colon of this limitation (hereinafter referred to as “alternative elements”). Dalcher discloses evaluating frames of a stack to determine if they have been overwritten (corresponding to “a base pointer is identified as being invalid” as well as “an invalid stack return address is identified”). EX1004, ¶[0024]; EX1002, ¶¶289-290,292,413. Dalcher teaches analyzing code preceding a return pointer in a frame which may indicate that the return pointer is invalid or overwritten (corresponding to “an invalid stack return

address is identified”). *Id.* Accordingly, Dalcher teaches at least two of the alternative elements of this limitation.

Pappas teaches monitoring of a memory stack (including a stack walk and memory checks) that is consistent with Dalcher and also teaches at least one of the alternative elements. EX1005, pp.6; EX1002, ¶¶161-162,165,172. Specifically, Pappas teaches performing a stack walk and other analysis to identify “abnormal control transfers that take place during [return-oriented programming] ROP code execution” (corresponding to “attempted execution of a return-oriented programming technique is detected”). EX1005, pp.3; EX1002, ¶¶103-120(citing EX1010),293-297,413.

Moreover, Dalcher and Pappas disclose “wherein an alert of suspicious behavior is triggered when the...memory check detects” the above-discussed alternative elements. **EX1002, ¶¶303-306.** For example, Dalcher discloses putting a system in quarantine when the above-discussed alternative elements are detected, which a PHOSITA would have understood to involve triggering an alert of suspicious behavior. **EX1004, ¶[0026]**Also, Pappas displays a visual alert of suspicious behavior when the above-discussed alternative element is detected. EX1002, ¶¶288-292, 296-297; EX1005, pp.10,14.

Therefore, Dalcher discloses this limitation and Pappas discloses compatible and consistent memory stack monitoring. EX1002, ¶¶292,297,299,421.

- g. wherein when an alert of suspicious behavior is triggered, the method further comprising preventing execution of a payload for the invoked function.**

Dalcher discloses monitoring to determine whether a compromise (suspicious behavior) occurred and a level of confidence in the determination, and if the level of confidence that a compromise has occurred is above a certain threshold, a flag is generated indicating the compromise (corresponding to “alert of suspicious behavior is triggered”). EX1004, ¶¶[0024]-[0026]; EX1002, ¶¶303-304. Dalcher discloses placing software or functions in quarantine that have been flagged as compromised (corresponding to “preventing ...from operating”). EX1004, ¶[0026]; EX1002, ¶304,413.

A PHOSITA would have understood that the quarantine process of Dalcher involves preventing operation and ultimately execution of quarantined software or functions (including payload for the invoked function). EX1002, ¶¶304. Pappas discloses terminating execution of code or frames (corresponding to “preventing execution of a payload for the invoked function”) that have been identified by memory analysis to be compromised/suspicious. EX1005, pp.455; EX1002, ¶¶305-306. A PHOSITA would have looked to Pappas for further compatible details on preventing execution of quarantined software or functions when a compromise (suspicious behavior) is detected. EX1002, ¶¶305-307,413.

Therefore, Dalcher and Pappas disclose this limitation. EX1002, ¶¶302,421.

For the foregoing reasons, a PHOSITA as of the earliest possible priority date would have found claim 9 of the '591 patent to be obvious in view of the combination of Dalcher and Pappas because these prior art publications disclose each and every limitation recited in the claim. *Id.* Accordingly, claim 9 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

10. Claim 10

The system according to claim 9, further comprising identifying an address associated with the invoked function; and when the address of the invoked function is found in a cache, and wherein the processor further executing an operation of preventing execution of a payload for the invoked function.

Claim 10 recites a limitation corresponding to the limitations recited in claim 2. EX1002, ¶413. As described above regarding claim 2, Dalcher and Pappas discloses this claim, and further discloses a processor for executing the method of claim 10. EX1002, ¶¶413,417,421,423.

Therefore, a PHOSITA would have found the limitations recited in claim 10 obvious in view of the Dalcher-Pappas combination. *Id.* As shown, the Dalcher-Pappas combination discloses all elements of claim 9 from which claim 10 depends. Accordingly, claim 10 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

11. Claim 11

The system according to claim 9, wherein the executing of the stack walk processing further comprises walking a stack frame associated with the invoked function from a low level user mode function to the originating address of the invoked function.

Claim 11 recites a limitation corresponding to the limitations recited in claim 3. EX1002, ¶413. As described above regarding claim 3, Dalcher and Pappas discloses this claim, and further discloses a system for executing the method of claim 11. EX1002, ¶¶413-414,415,421,423.

Therefore, a PHOSITA would have found the limitations recited in claim 11 obvious in view of the Dalcher-Pappas combination. EX1002, ¶¶414. As shown, the Dalcher-Pappas combination discloses all elements of claim 9 from which claim 11 depends. Accordingly, claim 11 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

12. Claim 12

The system according to claim 9, wherein the monitoring further comprises monitoring functions associated with memory management in the memory space by a user mode function.

Claim 12 recites a limitation corresponding to the limitations recited in claim 4. EX1002, ¶413. As described above regarding claim 4, Dalcher and Pappas discloses this claim, and further discloses a system for executing the method of claim 12. EX1002, ¶¶413-414,415,421,423.

Therefore, a PHOSITA would have found the limitations recited in claim 12 obvious in view of the Dalcher-Pappas combination. EX1002, ¶¶413-414,421. As shown, the Dalcher-Pappas combination discloses all elements of claim 9 from which claim 12 depends. Accordingly, claim 12 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

13. Claim 13

The system according to claim 9, wherein the monitoring further comprises applying, upon detecting a function call of one of the plurality of functions in the monitored memory space, a hooking technique to hook a lowest level user mode function of the invoked function, and wherein the executing of the stack walk processing executes a reverse stack walk from the lowest level user mode function of the invoked function.

Claim 13 recites a limitation corresponding to the limitations recited in claim 5. EX1002, ¶413. As described above regarding claim 5, Dalcher and Pappas discloses this claim, and further discloses a system for executing the method of claim 13. EX1002, ¶¶413-414,415,421,423.

Therefore, a PHOSITA would have found the limitations recited in claim 13 obvious in view of the Dalcher-Pappas combination. EX1002, ¶¶413-414,421. As shown, the Dalcher-Pappas combination discloses all elements of claim 9 from which claim 13 depends. Accordingly, claim 13 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

14. Claim 14

The system according to claim 9, wherein the plurality of functions comprises an NtAllocateVirtualMemory function, an NtProtectVirtualMemory function, an NtCreateProcess function, an NtCreateProcessEX function, an NtCreateUserIProcess function, and an LdrLoadDll function.

Claim 14 recites a limitation corresponding to the limitations recited in claim 6. EX1002, ¶413. As described above regarding claim 6, Dalcher and Pappas discloses this claim, and further discloses a system for executing the method of claim 14. EX1002, ¶¶413-414,415,421,423.

Therefore, a PHOSITA would have found the limitations recited in claim 14 obvious in view of the Dalcher-Pappas combination. *Id.* As shown, the Dalcher-Pappas combination discloses all elements of claim 9 from which claim 14 depends. Accordingly, claim 14 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

15. Claim 15

The system according to claim 13, wherein performing of the memory check comprises evaluating state information associated with a cache entry for the hooked function.

Claim 15 recites a limitation corresponding to the limitations recited in claim 7. EX1002, ¶413. As described above regarding claim 7, Dalcher and Pappas

discloses this claim, and further discloses a system for executing the method of claim 15. EX1002, ¶¶413-414,415,421,423.

Therefore, a PHOSITA would have found the limitations recited in claim 15 obvious in view of the Dalcher-Pappas combination. *Id.* As shown, the Dalcher-Pappas combination discloses all elements of claims 9 and 13, from which claim 15 depends. Accordingly, claim 15 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

16.Claim 16

The system according to claim 15, wherein when an address of the invoked function is found in a cache, and wherein the processor further executing an operation of preventing execution of a payload for the invoked function.

Claim 16 recites a limitation corresponding to the limitations recited in claim 8. EX1002, ¶413. As described above regarding claim 8, Dalcher and Pappas discloses this claim, and further discloses a system and processor for executing the method of claim 16. EX1002, ¶¶413-414,415,421,423.

Therefore, a PHOSITA would have found the limitations recited in claim 16 to be obvious in view of the Dalcher-Pappas combination. *Id.* As shown, the Dalcher-Pappas combination discloses all elements of claims 9, 13 and 15, from which claim 16 depends. Accordingly, claim 16 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

17. Claim 17

- a. A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:**

Dalcher discloses RAM and a CPU connected as the computing system of FIG. 2, in which the CPU executes instructions such as those that disclose the processes of the remaining limitations of Claims 17 and 18. EX1004, ¶¶[0012], claim-19; EX1002, ¶¶415-417, 425, 427.

Therefore, Dalcher discloses this limitation. *Id.*

- b. 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;**

Claim limitation 17[b] recites a limitation corresponding to the limitations recited in claim limitations 9[c] and 1[b] above. EX1002, ¶425. As described above regarding claim limitations 9[c] and 1[b], Dalcher and Pappas disclose this claim, and further disclose a system for executing the method of claim 17[b]. EX1002, ¶¶425, 428.

Therefore, Dalcher and Pappas disclose this limitation. *Id.*

- c. invoking one of the plurality of functions as a result of receiving a call from an application programming instance;**

Claim limitation 17[c] recites a limitation corresponding to the limitations recited in claim limitations 9[d] and 1[c] above. EX1002, ¶425. As described above regarding claim limitations 9[d] and 1[c], Dalcher and Pappas disclose this claim, and further disclose a system for executing the method of claim 17[c]. EX1002, ¶¶425,428.

Therefore, Dalcher in view of Pappas discloses this limitation. *Id.*

d. executing stack walk processing upon the invocation of one of the plurality of functions in the monitored memory space; and

Claim limitation 17[d] recites a limitation corresponding to the limitations recited in claim limitations 9[e] and 1[d] above. EX1002, ¶425. As described above regarding claim limitations 9[e] and 1[d], Dalcher and Pappas disclose this claim, and further disclose a system for executing the method of claim 17[d]. EX1002, ¶¶425,428.

Therefore, Dalcher and Pappas disclose this limitation. *Id.*

- e. 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, a return address is detected as being inside a virtual memory area, and a low level user mode function is attempted to be executed in a cache, wherein the cache is evaluated for an address of the invoked function,**

Claim limitation 17[e] recites elements of a limitation corresponding to the limitations recited in claim limitations 9[f] and 1[e]. EX1002, ¶425. As described above regarding claim limitations 9[f] and 1[e], Dalcher and Pappas disclose this claim limitation, and further discloses a system for executing the method of claim 17[e]. EX1002, ¶¶425,428.

The additional alternative element of this claim limitation relative to claim limitations 9[f] and 1[e] need not be taught to render the limitation obvious. *See e.g., Titanium Metals Corp.*, 778 F.2d at 782 (Fed. Cir. 1985) .

Nevertheless, Dalcher in view of Pappas teaches the feature of “wherein an alert of suspicious behavior is triggered when the performing of the memory check detects...a low level user mode function is attempted to be executed in a cache, wherein the cache is evaluated for an address of the invoked function.” EX1002,

¶433. Dalcher teaches the features of the alert of suspicious behavior as described regarding claim limitations 1[e]-[f] and teaches the features of evaluating an address of the invoked function in cache as described regarding claim limitations 2[a]-[b]. EX1002, ¶¶434-435. Further, Dalcher teaches analyzing code for a preceding call that explains invocation. EX1004, ¶[0039]; EX1002, ¶¶443-445. Dalcher also teaches freed memory to preserve at least a portion of the call stack, which a PHOSITA would have understood to be a cache. EX1002, ¶¶447-450; EX1004, ¶¶[0037]-[0038],[0046]; EX1007, pp.15-16; *see also* discussions regarding Claim 3 (discussing execution of and stack walk/memory checks relating to low level user mode functions).

Therefore, Dalcher discloses this limitation and Pappas discloses compatible and consistent memory stack monitoring. EX1002, ¶¶451-455. A PHOSITA would have understood that these disclosures of Dalcher-Pappas teach the features of “an alert of suspicious behavior is triggered when the performing of the memory check detects...a low level user mode function is attempted to be executed in a cache, wherein the cache is evaluated for an address of the invoked function.” EX1002, ¶¶457-464.

- f. wherein when an alert of suspicious behavior is triggered, preventing execution of a payload for the invoked function.**

Claim limitation 17[f] recites a limitation corresponding to the limitations recited in claim limitations 9[g] and 1[f]. EX1002, ¶425. As described above regarding claim limitations 9[g] and 1[f], Dalcher and Pappas disclose this claim limitation, and further disclose a system for executing the method of claim 17[f]. EX1002, ¶¶425,428,466.

For the foregoing reasons, a PHOSITA as of the earliest possible priority date would have found claim 17 of the '591 patent to be obvious in view of the combination of Dalcher and Pappas because these prior art publications disclose each and every limitation recited in the claim. *Id.* Accordingly, claim 17 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

18. Claim 18

- a. The computer-readable storage device according to claim 17, wherein performing of the memory check to determine if code execution is attempted from non-executable memory, the base pointer is identified as being invalid, an invalid stack return address is identified, execution of a return-oriented programming technique is attempted, the base pointer is detected as being outside a current thread stack, and the return address is detected as being inside a virtual memory area, occurs before an address of the invoked function is reached, and**

As described regarding claim limitations 1[e] and 1[f], Dalcher discloses performing memory analysis or stack analysis of an execution flow of a function and handling of code execution (corresponding to “performing a memory check”). EX1004, ¶¶[0023],[0027]; EX1002, ¶¶470-472. In particular, Dalcher teaches at

least two of these alternative elements of this limitation, which is a subset of the alternative elements of the Claim 18 quoted above.

For example, first, Dalcher discloses evaluating frames of a stack to determine if they have been overwritten (corresponding to “a base pointer is identified as being invalid” as well as “an invalid stack return address is identified”). EX1004, ¶¶[0024]; EX1002, ¶¶[472-473]. Second, Dalcher teaches analyzing code preceding a return pointer in a frame which may indicate that the return pointer is invalid or overwritten (corresponding to “an invalid stack return address is identified”). *Id.*

As described regarding claim limitation 1[c], Dalcher also discloses that the analysis of execution flow and handling of test events can be performed by additional monitoring after a function is invoked. EX1004, ¶¶[0023],[0033]; EX1002, ¶¶[227-232]. An invoked function may invoke further functions which are placed on the call stack and Dalcher discloses that these further functions may be analyzed before the originally invoked function is reached or executed (corresponding to “occurs before an address of the invoked function is reached”). EX1004, ¶¶[0023],[0033]; EX1002, ¶¶[270-289]; *see also* discussions regarding limitations 1[d]-1[f].

Further, Pappas teaches multiple levels of functions in user space of the operating system. EX1005, pp.450-451; EX1002, ¶¶[474-475]. Specifically, Pappas

discloses that an application may call a function which calls a Windows API function which then generates a system call to a function. *Id.* The system calls are cited as monitoring points in Pappas to ensure that the system calls were properly called by the invoked function through the Windows API. *Id.* Accordingly, the analysis of the system call and its origination is performed before the original invoked function in the stack is reached (corresponding to “occurs before an address of the invoked function is reached”). EX1005, pp.5-6; EX1002, ¶¶474-476; *see also* discussions regarding limitations 1[d]-1[f].

Therefore, Dalcher and Pappas disclose this limitation. *Id.*

b. wherein the performing of the memory check to determine whether the low level user mode function is attempted to be executed in the cache occurs upon reaching the address of the invoked function in the execution of the stack walk processing.

As noted regarding claim 17[e], Dalcher teaches performing memory analysis (corresponding to “performing of the memory check”) and performing a stack walk (corresponding to “execution of the stack walk processing”) as part of that analysis. EX1004, ¶¶[0019]-[0021],[0036]; EX1002, ¶480. Dalcher also teaches generating test events to further monitor execution flow and handling of code by the operating system (corresponding to “low level user mode function”). EX1004, ¶¶[0023]-[0027]; EX1002, ¶481. Dalcher also discloses monitoring freed memory to identify attempted execution in freed memory, which a PHOSITA would have understood to be identifying whether a “function is attempted to be

executed in the cache” as recited in this limitation. EX1004, ¶¶[0021]-[0026]; EX1002, ¶¶480-481.

Further, Pappas teaches multiple levels of functions in user space of the operating system. EX1005, pp.5-6; EX1002, ¶¶483-484. Pappas teaches analyzing call stacks (corresponding to “performing of the memory check”) and caching one or more function calls and their addresses via a process called last branch recording (LBR). EX1005, pp.3-6; EX1002, ¶¶483-484. Pappas then analyzes these stored branches (in modified form) by searching for an address corresponding to the target address about to be executed (corresponding to “occurs upon reaching the address of the invoked function in the execution of the stack walk processing.”). EX1005, pp.10; EX1002, ¶¶480-483. Pappas discloses hooking system calls and then performing a stack walk to ensure that the system call was properly called based on the information cached by LBR (corresponding to “determine whether the low level user mode function is attempted to be executed in the cache”). EX1005, pp.3-10; EX1002, ¶¶480-485.

For these reasons, the combination of Dalcher and Pappas disclose this limitation. EX1002, ¶485.

Moreover, a PHOSITA would have been motivated to configure Dalcher-Pappas’ process such that the memory check to determine “whether the low level user mode function is attempted to be executed in the cache occurs upon reaching

the address of the invoked function in the execution of the stack walk processing.”

EX1002, ¶481.

Therefore, a PHOSITA would have found the limitations recited in claim 18 obvious in view of the Dalcher-Pappas combination. *Id.* As shown, the Dalcher Pappas combination discloses all elements of claim 17, from which claim 18 depends. Accordingly, claim 18 should be found invalid as unpatentable under 35 U.S.C. § 103 based on Dalcher in view of Pappas.

VI. NO BASIS FOR DISCRETIONARY DENIAL

There is no basis to deny institution under 35 U.S.C. § 325(d).

There is parallel court litigation, so *Fintiv* factors are addressed preliminarily. First, no stay has been entered in the parallel litigation. Second, trial is currently set for August 19, 2024, so there is time to complete inter parte review. Third, it is early in the parallel litigation as fact discovery has not started. Fourth, Petitioner has identified the Dalcher prior art in Defendant’s invalidity contentions filed in the parallel litigation. Finally, Petitioner and Webroot are parties in the parallel litigation as identified in the Mandatory Notices.

Discretionary denial is strongly disfavored here because the “same or substantially the same art or arguments” were not presented to the Office, and the Office “erred in a manner material” to patentability by not asserting the Dalcher and Pappas references. *See Advanced Bionics, LLC v. MED-EL*

Elektromedizinische Gerate GmbH, IPR2019-01469, Paper 6, 8-9 (Feb. 13, 2020).

Analysis of this issue is guided by the “*Becton*” factors. *Id.* at 9-11.

Dalcher and Pappas were not cited, individually or in combination, in any of the Office communications. *See* EX1006. Regarding *Becton* factors (a) (material differences between asserted art and prior art) and (b) (cumulative nature of the asserted art and prior art), Dalcher and Pappas are materially different from the combination of references asserted in Office communications. Thus, the individual references of Dalcher and Pappas are not merely cumulative of references the Office relied. Regarding *Becton* factor (d) (extent of overlap), the arguments in this Petition do not overlap with arguments made during examination.

Even if the Board finds that the “same or substantially the same” art or arguments were presented to the Office, discretionary denial under § 325(d) is still not proper because the Office made a “material error” in not considering the Dalcher and Pappas references.

Regarding *Becton* factors (c) (evaluation of art during examination), neither of Dalcher and Pappas were used in any claim rejections, weighing strongly against exercising discretion. *See Cellco Partnership v. Huawei Device Co.*, IPR2020-01117, Paper 10, 13 (Feb. 3, 2021).

Regarding *Becton* factor (e) (examiner error) and (f) (other facts), this petition specifically points out and explains how Dalcher and Pappas discloses all limitations of the claims.

VII. CONCLUSION

Inter partes review of claims 1-18 of U.S. Patent No. 10,284,591 is requested based on the grounds described above.

Respectfully submitted,

The Marbury Law Group, PLLC

A handwritten signature in black ink, reading "Robert M. Hansen", written over a horizontal line.

Robert M. Hansen, Reg. No. 43,656

Scott B. Amankwatia, Reg. No. 78,226

Matthew J. Anderson, Reg. No. 78,257

Attorneys for Petitioner

Date: March 7, 2023

The Marbury Law Group, PLLC
11800 Sunrise Valley Drive
15th Floor
Reston, VA 20191
Tel: (703) 391-2900

CERTIFICATION OF SERVICE (37 C.F.R. §§ 42.6(e), 42.105(a))

The undersigned hereby certifies that the above-captioned **PETITION FOR INTER PARTES REVIEW OF U.S. PATENT NO. 10,284,591**, Petitioner's power of attorney, this petition for *inter partes* review, and all associated exhibits were served in their entireties on March 7, 2023, on the following party via FedEx to the Patent Owner by serving to the correspondence address of record for the '591 patent:

**Sprinkle IP Law Group
1301 W. 25th Street, Suite 400
Austin, TX 78705**


Robert M. Hansen, Reg. No. 43,656
Attorney for Petitioner

Date: March 7, 2023

The Marbury Law Group, PLLC
11800 Sunrise Valley Drive
15th Floor
Reston, VA 20191
Tel: (703)-391-2900

CERTIFICATE OF COMPLIANCE WITH TYPE-VOLUME LIMITATION

Pursuant to 37 C.F.R. § 42.24 *et seq.*, the undersigned certifies that this Petition complies with the 14,000-word type-volume limitation. The Petition contains **13,422** words, excluding the table of contents, table of authorities, exhibit list, mandatory disclosures, signature block, certificates and attached exhibits of the Petition.

The Marbury Law Group, PLLC

A handwritten signature in black ink, reading "Robert M. Hansen", written over a horizontal line.

Robert M. Hansen, Reg. No. 43,656
Attorney for Petitioner

Date: March 7, 2023

The Marbury Law Group, PLLC
11800 Sunrise Valley Drive
15th Floor
Reston, VA 20191
Tel: 703-391-2900