UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE PATENT TRIAL AND APPEAL BOARD

_____

FORTINET INC.
Petitioner

v.

SOPHOS LLC
Patent Owner

_____

INTER PARTES REVIEW OF U.S. PATENT NO. 8,261,344
Case IPR: _____

_____

PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 8,261,344 UNDER 35 U.S.C §§ 311-319 AND 37 C.F.R. §§ 42.1-.80 & 42.100-.123

Mail Stop: Patent Board
Patent Trial and Appeal Board
U.S. Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

TABLE OF CONTENTS

TABLE OF CASES

EXHIBIT LIST

No.	Exhibit
Ex. 1001	U.S. Patent No. 8,261,344 to Godwood et al.
Ex. 1002	The prosecution history of U.S. Patent No. 8,261,344
Ex. 1003	U.S. Patent No. 7,913,305 to Bodorin et al. ("Bodorin")
Ex. 1004	U.S. Patent No. 7,373,644 to Kissel ("Kissel")
Ex. 1005	U.S. Patent No. 5,951,698 to Chen et al. ("Chen")
Ex. 1006	U.S. Patent No. 7,448,084 to Apap et al. ("Apap")
Ex. 1007	Expert Declaration of Seth Nielson
Ex. 1008	Joint Claim Construction and Prehearing Statement in *Fortinet, Inc. v. Sophos, Inc.*, Case No. 3:13-cv-05831-EMC (N.D. Cal.)
Ex. 1009	Exhibit E to Sophos Inc. and Sophos LTD's Disclosure Of Asserted Claims and Infringement Contentions and Document Production Pursuant to Patent L.R. 3-1 And 3-2 in *Fortinet, Inc. v. Sophos, Inc.*, Case No. 3:13-cv-05831-EMC (N.D. Cal.)
Ex. 1010	U.S. Patent No. 8,713,686 to Kane ("Kane")
Ex. 1011	December 15, 2014 Transcript of Claim Construction Hearing in *Fortinet, Inc. v. Sophos, Inc.*, Case No. 3:13-cv-05831-EMC (N.D. Cal.)
Ex. 1012	Fred Cohen, Computer Viruses, Theory and Experiments, Computers & Security 6, at 22-35 (1986).
Ex. 1013	David M. Chess and Steve R. White, An Undetectable Computer Virus, IBM Thomas J. Watson Research Center, 2000.
Ex. 1014	Virus Bulletin, March 1991.
Ex. 1015	Thomas M. Chen, The Evolution of Viruses and Worms, Dept. of Electrical Engineering, SMU, 2004.
Ex. 1016	Virus Bulletin, November 1991.
Ex. 1017	Virus Bulletin, December 1991.

Ex. 1018	Virus Bulletin, November 1993.
Ex. 1019	Dmitry O. Gryaznov, Scanners of the Year 2000: Heuristics, Virus Bulletin Conference, at 225-234, Sept. 1995.
Ex. 1020	Virus Bulletin, September 1995.
Ex. 1021	Virus Bulletin, April 1990.
Ex. 1022	Virus Bulletin, February 2000.

I. INTRODUCTION

Fortinet Inc. ("Fortinet" or "Petitioner") respectfully requests *inter partes* review ("IPR") of Claims 1, 2, 10, 13, 16, and 17 of U.S. Patent No. 8,261,344 ("the '344 Patent"), Exhibit ("Ex.") 1001, under 35 U.S.C. §§ 311-319, 37 C.F.R. §§ 42.1-.80 and 37 C.F.R. §§ 42.100-.123. Generally, the '344 Patent discloses classifying and identifying malware and virus software by extracting certain features or characteristics of the software referred to in the '344 Patent as "genes." Ex. 1001 at Abstract. The disclosed methods then compare these extracted genes to a library of known genes. Ex. 1001 at 5:51-56. Once identified, the user may be notified of the malware and virus software. Ex. 1001 at 6:1-8. In addition to reporting, the malware may be blocked or removed. Ex. 1001 at 6:8-10.

As explained below, the Patent Owner obtained allowance of the claims by principally relying on specific and narrow definitions of "genes," library of "genes," as well as "functional blocks" as recited in the claims. More specifically, during the prosecution of the '344 Patent, the Patent Owner made it clear that the '344 Patent was patentable over the cited prior art because the '344 Patent disclosed that "[t]he genes are matched against functional blocks of code and the functionality is determined by matching sequences of API calls and the strings that are references." Ex. 1002 at 128. The Examiner in turn made it clear that these specific definitions of "genes" and "functional blocks" were critical in allowing the

claims. In the Notice of Allowance, the Examiner was clear that the aspect of the claimed invention that was not found in the prior art was "identifying one or more genes each describing one or more of the at least one functional block and the at least one property of the software as a sequence of APIs and strings." Ex. 1002 at 14.

However, in the pending litigation in the Northern District of California, the Patent Owner has broadly construed "gene" as "a piece of functionality or property of a program" and "functional block" as "a segment of the program that illustrates the function and execution flow of the program." The Patent Owner contends that its construction for "gene" is supported at 5:32-33 and 5:37-41 in the specification. The Paten Owner also contends that its construction for "functional block" is supported at 5:18-20, 5:27-28, and 5:32-41 in the specification. Under such broad claim interpretations, many prior art references disclose analyzing certain functionality or property of potential malware and identifying malware by comparing the analyzed functionality or property with a library of known malware. These prior art references render the claims of the '344 Patent invalid.

As set forth below, Claims 1, 2, 10, 13, 16, and 17 of the '344 Patent are invalid in light of prior art not cited during the prosecution of the '344 Patent. Elements of these claims were well known and widely used in the art well before the '344 Patent's purported priority date of June 30, 2006.

Identifying "functional blocks": Independent Claim 1 of the '344 patent recites identifying at least one functional block and at least one property of the software. However, as explained below, under the Patent Owner's broad construction of a "functional block" as "a segment of the program that illustrates the function and execution flow of the program," prior art not cited during prosecution demonstrates that this was also well known in the art. For example, U.S. Patent No. 7,913,305 to Bodorin et al. discloses identifying "behaviors," which are functionalities of the examined software. Ex. 1003 at 4:58-61. U.S. Patent No. 7,373,664 to Kissel discloses identifying "features," which include tokenized representations of the parsed script software where individual tokens and groups of tokens represent functions and execution flow of the program. Ex. 1004 at 3:43-61. As another example, U.S. Patent No. 5,951,698 to Chen et al. discloses identifying "instructions," which are segments of the macro that illustrate the function and execution flow of the macro. Ex. 1005 at 5:16-37.

Identifying "genes": Independent Claims 1 and 16 of the '344 patent recite identifying genes that describe a functional block and a property of the software. As Dr. Nielson explained, "genes" is not a term of art ordinarily used in computer science. Ex. 1007 at ¶ 42. Again, during the prosecution of the '344 Patent, the Examiner stated that this limitation was the reason for allowance. Ex. 1002 at 14. However, as explained below, prior art not cited during prosecution demonstrates

that this was also well known in the art. As Dr. Nielson explained, the concepts in the '344 Patent that relate to the term "genes," especially under this broad construction, are not new. Ex. 1007 at ¶ 42. For example, U.S. Patent No. 7,913,305 to Bodorin et al. discloses identifying a subset of the "behaviors" that are "interesting." These "interesting behaviors" are pieces of functionality of the software. Ex. 1003 at 4:51-5:20. The "features" disclosed in U.S. Patent No. 7,373,664 to Kissel may include keywords extracted from tokenized representation of script software that are pieces of functionality or property of the software. Ex. 1004 at 5:20-25. As another example, U.S. Patent No. 5,951,698 to Chen et al. discloses using instruction identifiers to identify functionality or one or more properties of the software that may indicate malware. Ex. 1005 at Fig. 6.

Providing a library of genes: Independent Claims 1 and 16 of the '344 patent recite providing a library of gene information. As explained in detail below, a library of software characteristics used to identify malware was well known in the art. For example, U.S. Patent No. 7,913,305 to Bodorin et al. discloses a "malware behavior signature store" that is a repository of behavior signatures of known malware. Ex. 1003 at 5:31-32. U.S. Patent No. 7,373,664 to Kissel discloses a "blocked feature instance library" that contains exemplars of feature instances that are indicative of malicious code. Ex. 1004 at 3:27-30. Likewise, U.S. Patent No. 5,591,698 to Chen et al. discloses a virus information module that

provides comparison data for the detection of viruses in macros. Ex. 1005 at 6:1-3. And as Dr. Nielson explained, scanning for malware by matching aspects of a particular piece of software against a library of those of known malware was a common way of detecting malware. Ex. 1007 at ¶¶ 28-30.

Reporting or blocking identified malware: Claims 1, 10, and 13 of the '344 Patent recite reporting to the user that malware has been found or blocking the malware from executing once that malware has been identified. As discussed above and explained by Dr. Nielson, anti-virus software was well known in the art by the priority date of the '344 Patent and it was common for anti-virus software to report or block identified malware. Ex. 1007 at ¶ 67-69.

Accordingly, the challenged claims of the '344 Patent should not have issued.

II. MANDATORY NOTICES

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

Fortinet is the real party-in-interest. Fortinet is located at 1090 Kifer Road, Sunnyvale, California 94086.

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

The '344 Patent is asserted in the following case in the Northern District of California: *Fortinet, Inc. v. Sophos, Inc. et al.,* 3:13-cv-05831-LB. Fortinet was first served with a counterclaim alleging infringement of the '344 patent on

January 24, 2014, and therefore is not barred to petition for *inter partes* review under 35 U.S.C. § 315(a)(3).

C. Lead and Backup Counsel and Service Information (37 C.F.R. §42.8(b)(3)-(4)).

The designations of counsel and addresses for service are:

LEAD COUNSEL	BACKUP COUNSEL
Jason Liu Reg. No. 51,955 jasonliu@quinnemanuel.com **Postal and Hand Delivery Address:** Quinn Emanuel Urquhart & Sullivan 50 California St, 22nd Floor San Francisco, CA 94111 Phone: 415.875.6434 Fax: 415.875.6700	Robert Kang Reg. No. 59,609 robertkang@quinnemanuel.com **Postal and Hand Delivery Address:** Quinn Emanuel Urquhart & Sullivan 50 California St, 22nd Floor San Francisco, CA 94111 Phone: 415.875.6318 Fax: 415.875.6700 John Neukom (to be admitted *pro hac vice*) johnneukom@quinnemanuel.com Jordan R. Jaffe (to be admitted *pro hac vice*) jordanjaffe@quinnemanuel.com **Postal and Hand Delivery Address:** Quinn Emanuel Urquhart & Sullivan 50 California St, 22nd Floor San Francisco, CA 94111 Phone: 415.875.6300 Fax: 415.875.6700

III. GROUNDS FOR STANDING

Petitioner certifies that the '344 Patent is available for *inter partes* review, and that Petitioner is not barred or estopped from requesting this review. Petitioner is filing Power of Attorney and an Exhibit List concurrently with this petition, as

well as all required fees. If any additional fees are due at any time through the course of the IPR, the undersigned authorizes the U.S. Patent and Trademark Office to charge such fees to Deposit Account No. 505708.

IV. IDENTIFICATION OF CHALLENGE AND RELIEF REQUESTED

Pursuant to 37 C.F.R. § 42.104(b)(2), *inter partes* review of the '344 patent is requested in view of the following references, each of which qualifies as prior art to the '344 patent under 35 U.S.C. § 102(a), (b), and/or (e):

a. U.S. Patent No. 7,913,305 to Bodorin et al. ("Bodorin") was filed on January 30, 2004, which is prior to the earliest filing date claimed by the '344 patent (June 30, 2006), and subsequently issued on March 22, 2011. Bodorin is therefore available as prior art under pre-AIA 35 U.S.C. § 102(e). Bodorin was not cited during the original prosecution of the '344 patent.

b. U.S. Patent No. 7,373,664 to Kissel ("Kissel") was filed on December 16, 2002, which is prior to the earliest filing date claimed by the '344 patent (June 30, 2006), and subsequently issued on May 13, 2008. Kissel is therefore available as prior art under pre-AIA 35 U.S.C. § 102(e). Kissel was not cited during the original prosecution of the '344 patent.

c. U.S. Patent No. 5,951,698 to Chen et al. ("Chen") issued on September 1999, which is more than a year prior to the earliest filing date claimed by the '344 patent (June 30, 2006). Chen is therefore available as prior art under

pre-AIA 35 U.S.C. § 102(b). Chen was not cited during the original prosecution of the '344 patent.

d. U.S. Patent No. 7,448,084 to Apap et al. ("Apap") was filed on January 27, 2003, which is prior to the earliest filing date claimed by the '344 patent (June 30, 2006), and subsequently issued on November 4, 2008. Apap is therefore available as prior art under pre-AIA 35 U.S.C. § 102(e). Apap was not cited during the original prosecution of the '344 patent.

Petitioner requests cancellation of challenged Claims 1, 2, 10, 13, 16 and 17 under the following statutory grounds:

A. Claims 1 and 2 are anticipated by Bodorin.

B. Claims 10 and 13 are rendered obvious by Bodorin.

C. Claims 1, 2, 10, and 13 are anticipated by Kissel.

D. Claims 16 and 17 are rendered obvious in light of Kissel further in view of Apap.

E. Claims 16 and 17 are rendered obvious in light of Bodorin further in view of Kissel and Apap.

F. Claims 1, 2, 10, and 13 are anticipated by Chen.

1. How the Claims Are Unpatentable.

Pursuant to 37 C.F.R. § 42.104(b)(4), an explanation of how the challenged claims of the '344 Patent are unpatentable under the statutory grounds identified

above, including the identification of where each element of the claim is found in the prior art, is provided in Section VII, *infra*. There is a reasonable likelihood that at least one claim of the '344 Patent is unpatentable under 35 U.S.C. §§ 102 and 103.

2. Supporting Evidence.

Pursuant to 37 C.F.R. § 42.104(b)(5), the exhibit numbers of the supporting evidence relied upon and the relevance of the evidence to the challenges raised, including identification of specific portions of the evidence that support the challenge, are provided herein. An Exhibit List identifying the exhibits is included. *See supra* at iv-v. In further support of the proposed grounds of rejection, this Petition is accompanied by the declaration of Dr. Seth Nielson. *See* Ex. 1007.

V. THE '344 PATENT AND PROSECUTION HISTORY

A. The '344 Patent.

The '344 Patent, titled "Method and System for Classification of Software Using Characteristics and Combinations of Such Characteristics," issued on September 4, 2012. The '344 patent originally filed on June 30, 2006.

The '344 Patent is directed to classifying software (*e.g.*, malicious software) based in part on "identifying one or more genes that appear in software." *See, e.g.*, Ex. 1001 at 2:31-50, 6:43-45. The '344 Patent distinguishes its solution from other malware detection techniques, including: "traditional malware protection

techniques . . . based around anti-virus vendors creating signatures for known malware and products that scan systems searching for those specific signatures," Ex. 1001 at 1:59-61; "[h]ueristic techniques [that] look at various properties of a file and not necessarily the functionality of the program," Ex. 1001 at 2:10-13; and techniques that "run[] malware and attempt[] to stop execution if malicious behavior is observer [sic] to happen," Ex. 1001 at 2:13-15.

The '344 Patent purports to offer a different approach, using what it calls "genes." To classify software and detect potential malware, the '344 Patent proposes the following basic steps: (1) extracting "functional blocks" from a software file, *see, e.g.*, Ex. 1001 at 5:17-28, 7:62-63; (2) searching the "functional blocks" for one or more "genes," Ex. 1001 at 5:29-45, 7:64-67; and (3) matching the identified "genes" against a "library" which includes "a number of classifications based on groupings of genes," Ex. 1001 at 5:46-50, 7:60-61, 8:1-4. Software is then classified based on this matching. *Id*. Fig. 2 of the '344 Patent illustrates the process of detecting malware:

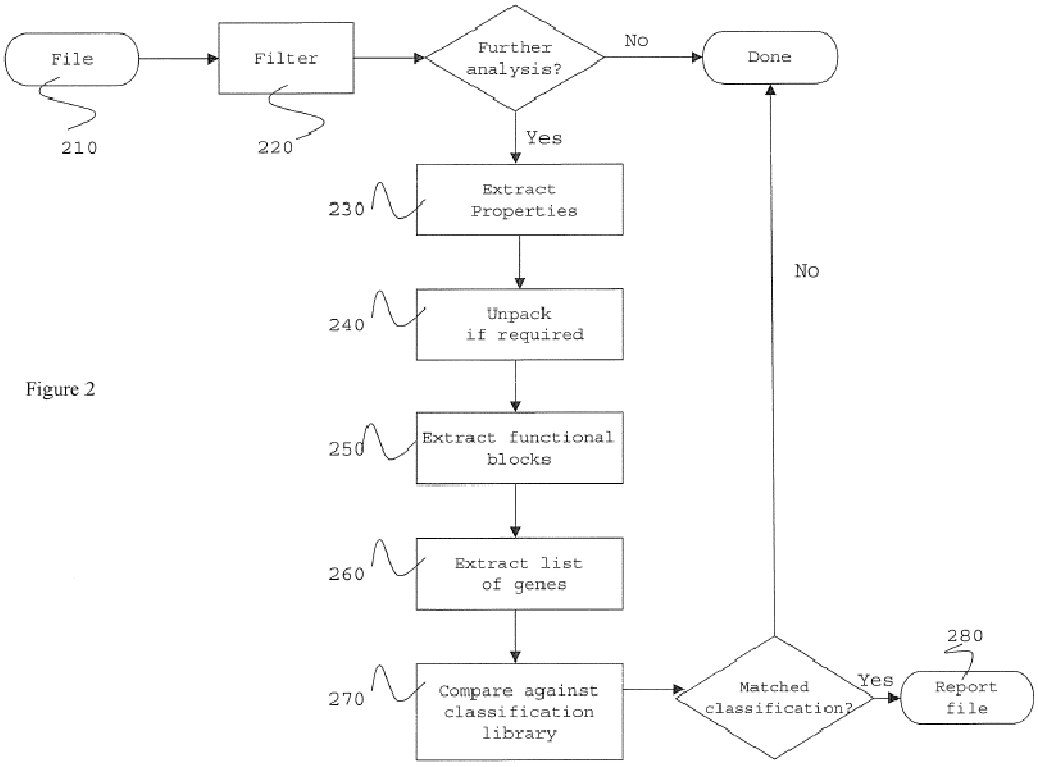


Figure 2

Ex. 1001 at Fig. 2.

B. Prosecution History.

On June 30, 2006, Patent Owner filed Application Serial No. 11/428,203 ("the '203 application"). Originally, 33 claims were pending in the '203 application. In a subsequent first Office Action dated May 17,2010, the Examiner rejected all pending claims under 35 U.S.C. § 101 and also under § 102(a) and (e) as being anticipated by U.S. Patent Publication No. 2006/0123244 to Gheorghescu et al. ("Gheorghescu"). Ex. 1002 at 163-173.

In response, Patent Owner filed an Amendment on January 24, 2011. Among other edits, Patent Owner amended Claims 1 and 16 to include the

limitation "using a processor." Patent Owner also added the limitation "notifying a user of said one or more classifications" to Claim 1 and "returning a result of said software classification based on said set of genes to a user" to Claim 16. Ex. 1002 at 120-21. Furthermore, Patent Owner argued Gheorghescu did not anticipate because Gheorghescu only disclosed a "level of scrutiny [that] is limited to the basic block [of program code], and individual elements of functionality are not directly analyzed." Ex. 1002 at 127-128. In contrast, the alleged invention of the '203 application disclosed "a gene [that] is a piece of functionality or property of the program" and "[t]he genes are matched against functional blocks of code and the functionality is determined by matching sequences of API calls and the strings that are references." *Id*.

In a subsequent final Office Action dated March 16, 2011, the Examiner maintained the § 102 rejection based on Gheorghescu. Patent Owner then filed a Request for Continued Examination, and amended independent Claims 1 and 16 to include the limitation "analyzing all functional elements of software," and "wherein a gene is a single piece of functionality or a property of software." Ex. 1002 at 85. As a result of the amendments to the claims, the Examiner withdrew the § 102 rejection based on Gheorghescu. Instead, in the Office Action dated October 4, 2011, the Examiner set forth a new rejection under § 103 in light of

U.S. Patent Application Publication No. 2008/0040710 to Chiriac and further in view of Gheorghescu. Ex. 1002 at 67-76.

Patent Owner amended the claims once again on April 3, 2012. Among other changes, Patent Owner amended independent Claims 1 and 16 to include limitations "providing a library of gene information including a number of classifications based on groupings of genes." Patent Owner also amended independent Claims 1 and 16 so that they recite that the genes extracted from the software describe a functional block and a property of the software as a sequence of APIs and strings. Ex. 1002 at 54.

The Examiner then allowed the claims. In the Notice of Allowance, the Examiner was clear that the aspect of the claimed invention that was not found in the prior art was "identifying one or more genes each describing one or more of the at least one functional block and the at least one property of the software as a sequence of APIs and strings." Ex. 1002 at 14. However, as demonstrated below, this limitation, as well as all other limitations of Claims 1, 2, 10, 13, 16, and 17 of the '344 Patent, were well known in the art before June 2006. Accordingly, these claims of the '344 Patent are invalid and should not have issued.

VI. CONSTRUCTION OF THE CHALLENGED CLAIMS

In an *inter partes* review, claim terms in an unexpired patent are interpreted according to their broadest reasonable interpretation ("BRI") in view of the

specification in which they appear. 37 C.F.R. § 42.100(b); Office Patent Trial Practice Guide, 77 Fed. Reg. 48,756, 48,766 (Aug. 14, 2012). The USPTO uses BRI because, among other reasons, the patentee has the opportunity to amend its claims in this proceeding. *See, e.g.,* Office Patent Trial Practice Guide, 77 Fed. Reg. 48,756, 48,764 (Aug. 14, 2012) ("Since patent owners have the opportunity to amend their claims during IPR, PGR, and CBM trials, unlike in district court proceedings, they are able to resolve ambiguities and overbreadth through this interpretive approach, producing clear and defensible patents at the lowest cost point in the system."). As required by the applicable rules, this Petition uses the BRI standard.

The BRI of claim terms here may be different from the construction that those same terms may receive following claim construction proceedings in district court. *See, e.g.*, *In re Trans Texas Holdings Corp.*, 498 F.3d 1290, 1297 (Fed. Cir. 2007). Thus the claim constructions presented in this Petition, including where Petitioner does not propose an express construction, do not necessarily reflect the claim constructions that Petitioner believes should be adopted by a district court under *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005).

In presenting this Petition, unless otherwise stated, the grounds set forth herein are based on (1) the proposed claim construction offered by the Patent Owner in *Fortinet, Inc. v. Sophos, Inc.*, Case No. 3:13-cv-05831-EMC (N.D. Cal.),

or (2) for terms where Patent Owner has not explicitly offered a claim construction, on Petitioner's understanding of Patent Owner's claim construction based upon Patent Owner's infringement contentions submitted in *Fortinet, Inc. v. Sophos, Inc.*, Case No. 3:13-cv-05831-EMC (N.D. Cal.). A claim construction order has not issued in this Northern District of California case. Petitioner does not concede that any claim constructions impliedly or expressly proposed by Patent Owner are appropriate for the district court litigation, where a different legal standard applies to the construction of the asserted claim terms.

Instead, Petitioner presents these proposed constructions to the Board for consideration in determining the BRI because, although Petitioner believes Patent Owner's constructions to be inconsistent with the intrinsic evidence, Patent Owner considers these constructions correct under *Phillips*, and therefore necessarily also considers them within the appropriate scope of the BRI. Petitioner further submits these constructions under 35 U.S.C. § 301(a)(2), which encourages submission of claim construction materials to prevent patentees from arguing broad constructions under *Phillips* while simultaneously arguing narrow constructions as the BRI.

Petitioner accordingly respectfully requests that the Board adopt the following constructions as the BRI of the limitations discussed below (reproduced and highlighted in claim 1 for the Board's reference):

1. A method for classifying software, said method comprising;

[a] providing a library of ***gene*** information including a number of classifications based on groupings of ***genes***;

[b] identifying at least one ***functional block*** and at least one property of the software;

[c] identifying one or more ***genes*** each describing one or more of the at least one ***functional block*** and the at least one property of the software as a sequence of APIs and strings;

[d] matching the one or more ***genes*** against one or more of the number of classifications using a processor;

[e] classifying the software based on the matching to provide a classification for the software; and

[f] notifying a user of the classification of the software.

"**gene**"— *a piece of functionality or property of a program.* In the pending litigation in the Northern District of California, the Patent Owner proposes to broadly construe "gene" as "a piece of functionality or property of a program." Ex. 1008 at 32. This construction is based on the portion of the specification that reads "[a] gene is a piece of functionality or property of a program." Ex. 1001 at 5:33-34. This is apparently motivated to support the Patent Owner's infringement contentions, where the Patent Owner has mapped any characteristic of a computer program, "such as file size, name, [and] type," as a "gene." Ex. 1009 at 5. During the claim construction hearing, counsel for the Patent Owner made it clear that the Patent Owner understood that "gene" can be a piece of functionality ***or*** property of

a program. Ex. 1011 at 116. Accordingly, the BRI applied herein for "gene" is "a piece of functionality or property of a program."

As explained in detail below, this limitation is disclosed by Bodorin, Kissel, and Chen. The "interesting behaviors" disclosed in Bodorin are pieces of functionality of the software. Ex. 1003 at 4:51-5:20. The "features" disclosed in Kissel may include keywords extracted from tokenized representation of script software that are pieces of functionality or property of the software. Ex. 1004 at 5:20-25. Additionally, Chen discloses using instruction identifiers to identify functionality or one or more properties of the software that may indicate malware. Ex. 1005 at Fig. 6.

"**functional block**"— *a segment of the program that illustrates the function and execution flow of the program*. In the pending litigation in the Northern District of California, the Patent Owner proposes to broadly construe "functional block" as "a segment of the program that illustrates the function and execution flow of the program." Ex. 1008 at 32. This construction is based on the portion of the specification that reads "[t]hese functional blocks include sequences of application program interface (API) calls, string references, etc., which illustrate the function and execution flow of the program." Ex. 1001 at 5:18-20. The Patent Owner's broad construction is consistent with its infringement contentions, where the Patent Owner has mapped attributes such as "virus signature, file type/pattern,

grayware, and heuristics" as "at least one functional block and at least one property of the software." Ex. 1009 at 16. Accordingly, the BRI applied herein for "functional block" is "a segment of the program that illustrates the function and execution flow of the program."

As explained in detail below, this limitation is disclosed by Bodorin, Kissel, and Chen. Bodorin discloses identifying "behaviors," which are functionalities of the examined software. Ex. 1003 at 4:58-61. Kissel discloses identifying "features," which may be tokenized representations of the script software where individual tokens and groups of tokens represent functions and execution flow of the program. Ex. 1004 at 3:43-61. Finally, Chen discloses identifying "instructions," which are segments of the macro that illustrate the function and execution flow of the macro. Ex. 1005 at 5:16-37.

VII. DETAILED EXPLANATION OF GROUNDS FOR INVALIDITY

As explained below pursuant to 37 C.F.R. § 42.104(b), the cited prior art anticipates or renders obvious the challenged claims of the '344 Patent. All of the references relied upon herein were not considered by the Patent Office in examination of the '344 Patent. These references disclose the features the Patent Owner argued were not in the prior art cited by the Examiner, including the "one or more genes each describing one or more of the at least one functional block and

the at least one property of the software as a sequence of APIs and strings," which were disclosed by Bodorin, Kissel, and Chen.

A. Claims 1 and 2 Are Anticipated by Bodorin

Bodorin discloses a malware detection system that determines whether an executable code module is malware according to behaviors exhibited by the code module while executing. Ex. 1003 at Abstract. A block diagram of the malware detection system is shown in Fig. 2.

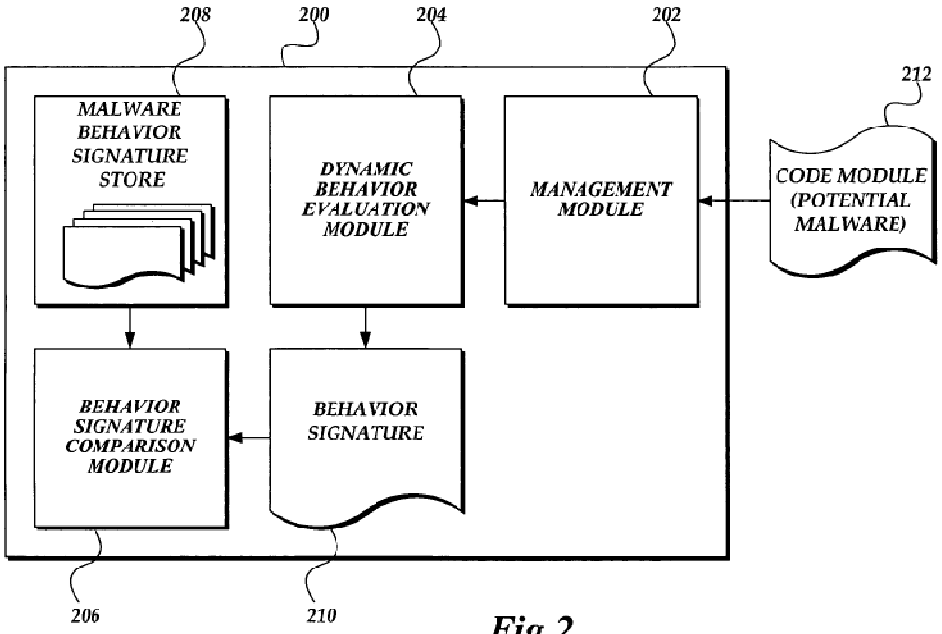


Fig.2.

Ex. 1003 at Fig. 2.

The management module 202 first evaluates the code module 212, which potentially may be malware, to determine the appropriate type of dynamic behavior evaluation module needed. Ex. 1003 at 4:26-29. Once determined, the

code module 212 is executed inside the dynamic behavior evaluation module 204 while the dynamic behavior evaluation module 204 records behaviors, including interesting behaviors that are potentially associated with malware, exhibited by the code module 212. Ex. 1003 at 4:39-58. Each dynamic behavior evaluation module represents a virtual environment in which the code module 212 runs. If code module 212 is malware, then the destructive behaviors of the code module 212 are confined to the virtual environment. Ex. 1003 at 4:39-50. Interesting behaviors include those shown in Table A, reproduced below. As shown in the table, the interesting behaviors include API calls such as RegisterServiceProcess() and strings such as application_name. Notably, these behaviors monitored in Bodorin include some of the very same API calls that are the bases for "genes" in the '344 Patent. *See, e.g.,* "CopyFileA" at Ex. 1003 at 5:10 and '344 Patent at 6:34.

TABLE A

RegisterServiceProcess ( )	ExitProcess ( )
Sleep ( )	GetCommandLine ( )
WinExec (application_name)	CreateProcessA (proces_name, parameters_list)
InternetOpenUrlA (URL_name)	GlobalAlloc ( )
InternetOpenA (URL_name)	GetTickCount ( )
IntenetClosetHandle ( )	CopyFileA (new_name, existing_name)
CreateFileA (file_name)	GetWindowsDirectoryA( )
ReadFile ( )	GetSystemDirectoryA ( )
WriteFile ( )	GetModuleFileNameA ( )
Close Handle ( )	LoadLibraryA (library_name)
RegOpenKeyA (key_name)	GetProcAddress (procedure_name)
RegSetValueA (subkey_name)	FindFirstFileA (file_specificator)
RegSetValuExA (subkey_name)	FindNextFileA ( )
RegCloseKey ( )	FindClose ( )
UrlDownloadToFileA (url_name, file_name)	

Ex. 1003 at 5:2-20.

The gene "CopySys" copies itself to a system folder and includes the following functions, for example:
GetSystemDirectoryA
GetModuleHandleA
GetModuleFileNameA
".exe"
CopyFileA
The gene "Runkey" sets a run key and includes, as an example, the following functions:
"software\microsoft\windows\currentversion\run"
RegCreateKeyExA
RegSetValueExA
The gene "Exec" executes other programs and includes, as an example, the following functions:
CreateProcessA
The three genes in this example form a classification defined by a group of the three genes, and may be used to classify the

Ex. 1001 at 6:28-44.

Bodorin collects recorded interesting behaviors into behavior signatures and they are illustrated in Figs. 5A and 5B:

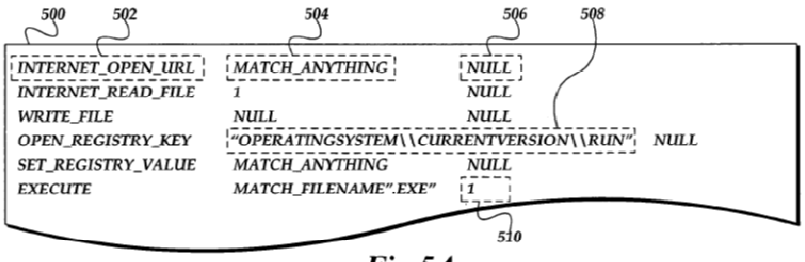


Fig.5A.

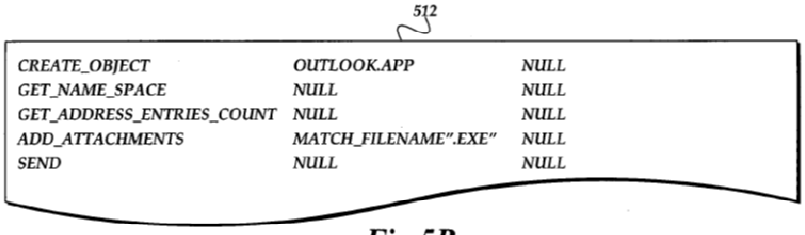


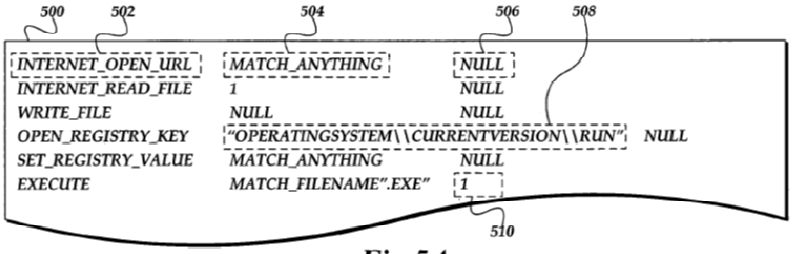
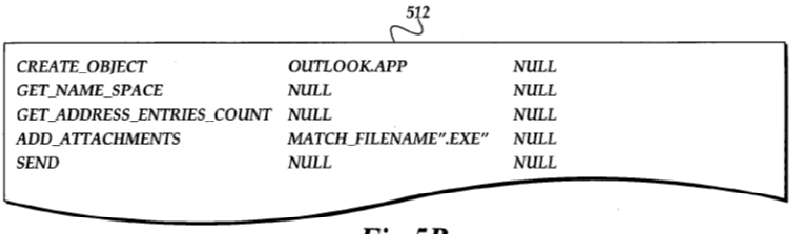
Fig.5B.

Ex. 1003 at Figs. 5A and 5B.

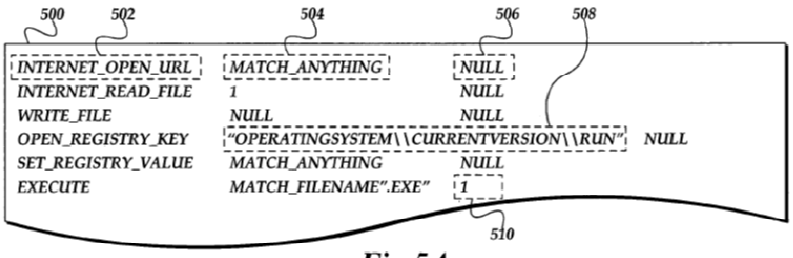
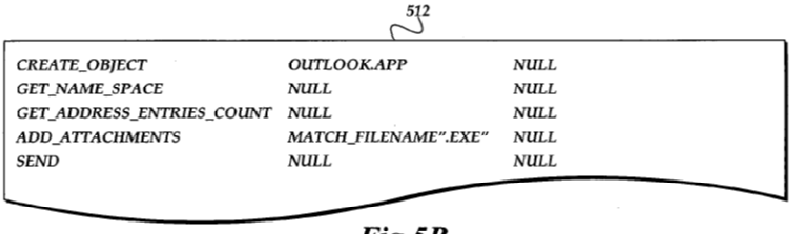
Once the behavior signature has been generated, the behavior signature comparison module 206 takes the behavior signature and compares it against behavior signatures of known malware that are stored in the malware behavior signature store 208. Ex. 1003 at 5:27-31. The malware behavior signature store 208 is a repository of behavior signatures of known malware. Ex. 1003 at 5:31-32. If the behavior signature comparison module 206 is able to completely match the behavior signature 210 against a known malware behavior signature in the malware behavior signature store 208, the malware detection system 200 will report that the code module is malware. Ex. 1003 at 5:44-49.

Each and every limitation of the Claims 1 and 2 of the '344 Patent are disclosed by Bodorin, in the same arrangement as in the '344 Patent, and are therefore anticipated.

Claim 1	**Bodorin**
A method for classifying software, said method comprising;	Ex. 1003 at Abstract: A malware detection system that determines whether an executable code module is malware according to behaviors exhibited while executing is presented. *See also* Ex. 1007 at ¶ 58.
[a] providing a library of gene information including a number of classifications based on groupings of genes;	Ex. 1003 at 5:27-43 (emphasis added): Once the behavior signature 210 has been generated, the behavior signature comparison module 206 takes the behavior signature and compares it against behavior signatures of known malware that are stored in the malware behavior signature store 208. ***The malware behavior signature store 208 is a repository of behavior signatures of known malware. However, unlike signature files of antivirus software 104, behavior signatures stored in the malware behavior signature store 208 are based on exhibited behaviors of malware.*** As such, even though all variable names and routine names within source code for malware are modified by a malicious party, the exhibited behaviors of the malware remain the same, and will be detected by the malware detection system 200 when the behavior signature comparison module 206 matches a code module's behavior signature 210 to a known malware's behavior signature in the malware behavior signature store 208.

	Ex. 1003 at 7:16-22: FIGS. 5A and 5B are block diagrams illustrating exemplary behavior signatures of hypothetical code modules, such as the behavior signature 210 described above. Each row of items in the exemplary behavior signatures, including behavior signature 500 and behavior signature 512, represents a behavior that was exhibited by a code module and recorded by the dynamic behavior evaluation module 204.  500 502 504 506 508 INTERNET_OPEN_URL MATCH_ANYTHING NULL INTERNET_READ_FILE 1 NULL WRITE_FILE NULL NULL OPEN_REGISTRY_KEY "OPERATINGSYSTEM\\CURRENTVERSION\\RUN" NULL SET_REGISTRY_VALUE MATCH_ANYTHING NULL EXECUTE MATCH_FILENAME".EXE" 1 510 Fig.5A.  512 CREATE_OBJECT OUTLOOK.APP NULL GET_NAME_SPACE NULL NULL GET_ADDRESS_ENTRIES_COUNT NULL NULL ADD_ATTACHMENTS MATCH_FILENAME".EXE" NULL SEND NULL NULL Fig.5B. *See also* Ex. 1007 at ¶¶ 59-61.
[b] identifying at least one functional block and at least one property of the software;	Ex. 1003 at 4:51-5:20: As the code module 212 executes within the dynamic behavior evaluation module 204, the dynamic behavior evaluation module records "interesting" behaviors exhibited by the code module. Interesting behaviors are those which a user or implementer of the malware detection system 200 has identified as interesting, potentially associated with malware, and are used to compare the behaviors of the code module 212 against known malware behaviors. Table A includes an exemplary, representative list of "interesting" behaviors, including parameters that are also considered interesting, that are recorded by a dynamic behavior evaluation module 204. It should be understood that these listed "interesting" behaviors are exemplary only, and should not be construed as limiting upon the present invention. Those skilled in the art will

	recognize that each individual type of code module may include a unique set of interesting behaviors. Additionally, as new features are added to each environment, or as computer systems change, additional behaviors may be added as "interesting," while others may be removed. TABLE A RegisterServiceProcess ( ) Sleep ( ) WinExec (application_name) InternetOpenUrlA (URL_name) InternetOpenA (URL_name) IntenetClosetHandle ( ) CreateFileA (file_name) ReadFile ( ) WriteFile ( ) Close Handle ( ) RegOpenKeyA (key_name) RegSetValueA (subkey_name) RegSetValuExA (subkey_name) RegCloseKey ( ) UrlDownloadToFileA (url_name, file_name) ExitProcess ( ) GetCommandLine ( ) CreateProcessA (proces_name, parameters_list) GlobalAlloc ( ) GetTickCount ( ) CopyFileA (new_name, existing_name) GetWindowsDirectoryA( ) GetSystemDirectoryA ( ) GetModuleFileNameA ( ) LoadLibraryA (library_name) GetProcAddress (procedure_name) FindFirstFileA (file_specificator) FindNextFileA ( ) FindClose ( ) *See also* Ex. 1007 at ¶ 62.
[c] identifying one or more genes each describing one or more of the at least one functional block and the at least one property of the software as a sequence of APIs and strings	Ex. 1003 at 7:16-22: FIGS. 5A and 5B are block diagrams illustrating exemplary behavior signatures of hypothetical code modules, such as the behavior signature 210 described above. Each row of items in the exemplary behavior signatures, including behavior signature 500 and behavior signature 512, represents a behavior that was exhibited by a code module and recorded by the dynamic behavior evaluation module 204.

	 ***Fig.5A.***  ***Fig.5B.*** Ex. 1003 at Claim 1 (emphasis added): . . . at least one dynamic behavior evaluation module, wherein each dynamic behavior evaluation module provides a virtual environment for executing a code module of a particular type, and wherein ***each dynamic behavior evaluation module records interesting API function calls that the code module makes as it is executed***, wherein the interesting API function calls are specified by a user and comprise a portion of all API function calls that the code module makes, wherein only the interesting API function calls, but not all API function calls, that the code module makes during execution in the dynamic behavior evaluation module are recorded into a behavior signature corresponding to the code module; *See also* Ex. 1007 at ¶ 63.
[d] matching the one or more genes against one or more of the number of classifications using a processor;	Ex. 1003 at 5:27-43 (emphasis added): ***Once the behavior signature 210 has been generated, the behavior signature comparison module 206 takes the behavior signature and compares it against behavior signatures of known malware that are stored in the malware behavior signature store 208***. The malware behavior signature store 208 is a repository of behavior signatures of known malware. However, unlike signature files of antivirus software 104, behavior signatures stored in the malware behavior signature store 208 are based on exhibited behaviors of malware. As such, even though all

	variable names and routine names within source code for malware are modified by a malicious party, the exhibited behaviors of the malware remain the same, and will be detected by the malware detection system 200 when the behavior signature comparison module 206 matches a code module's behavior signature 210 to a known malware's behavior signature in the malware behavior signature store 208. *See also* Ex. 1007 at ¶ 64.
[e] classifying the software based on the matching to provide a classification for the software; and	Ex. 1003 at 5:44-49: According to aspects of the present invention, if the behavior signature comparison module 206 is able to completely match the behavior signature 210 against a known malware behavior signature in the malware behavior signature store 208, the malware detection system 200 will report that the code module is malware. *See also* Ex. 1007 at ¶ 64.
[f] notifying a user of the classification of the software.	Ex. 1003 at 5:44-49 (emphasis added): According to aspects of the present invention, if the behavior signature comparison module 206 is able to completely match the behavior signature 210 against a known malware behavior signature in the malware behavior signature store 208, ***the malware detection system 200 will report that the code module is malware***. *See also* Ex. 1007 at ¶ 65.

Claim 2	**Bodorin**
The method of claim 1, wherein the classification for the software includes malware.	Ex. 1003 at 5:44-49: According to aspects of the present invention, if the behavior signature comparison module 206 is able to completely match the behavior signature 210 against a known malware behavior signature in the malware behavior signature store 208, the malware detection system 200 will report that the code module is malware.

	See also Ex. 1007 at ¶ 64.

B. Claims 10 and 13 Are Rendered Obvious in View of Bodorin Itself

Claims 10 and 13 of the '344 Patent recite limitations that relate to removing and blocking access to identified malware. Although the malware detection system disclosed in Bodorin does not include functionality that removes and blocks access to identified malware, it would have been obvious to a person of ordinary skill in the art at the time of the '344 priority date, in view of Bodorin itself, to add such functionality. This is because there is explicit teaching, suggestion, and motivation in Bodorin to combine. Bodorin explains that existing anti-virus software known in the art includes functionality to remove or block access to known malware. Ex. 1003 at 1:44-61. Bodorin then explicitly states that it would be advantageous to combine the dynamic malware detection system it discloses with prior art anti-virus software. Ex. 1003 at 3:58-4:12. Specifically, Bodorin states "while the malware detection system may be used as a stand-alone product, it may also be used in conjunction with current anti-virus software." Ex. 1003 at 4:2-4. Bodorin further explains that "when the present invention is used in combination with current anti-virus software, it may be beneficial to use the anti-virus software's signature matching techniques as a first step in securing a computer from malware, before turning to the malware detection system described herein." Ex. 1003 at 4:8-12. Accordingly, one of ordinary skill reading Bodorin

would be motivated to combine its malware detection system with prior art anti-virus software, disclosed in Bodorin itself, that removes and blocks access to identified malware. *See* Ex. 1007 at ¶¶ 66-69.

The following claim charts demonstrate, on a limitation-by-limitation basis, how Claims 10 and 13 are rendered obvious by Bodorin alone.

Claim 10	**Bodorin**
The method of claim 1, further comprising removing the software when the software is classified as malware or unwanted software.	Ex. 1003 at 1:44-61 (emphasis added): As shown in FIG. 1A, a malware 102 is directed over a network 106 to the computer 110, as indicated by arrow 108. It will be appreciated that the malware 102 may be directed to the computer 110 as a result of a request initiated by the computer, or directed to the computer from another network device. However, as mentioned above, before the known malware 102 reaches the computer 110, anti-virus software 104 installed on the computer intercepts the malware and examines it. As is known in the art, currently, anti-virus software scans the incoming data as a file, searching for identifiable patterns, also referred to as signatures, associated with known-malware. ***If a malware signature is located in the file, the anti-virus software 104 takes appropriate action, such as deleting the known malware/infected file, or removing the malware from an infected file, sometimes referred to as cleaning the file.*** In this manner, anti-virus software 104 is able to prevent the known malware 102 from infecting the computer 110, as indicated by the arrow 112. *See also* Ex. 1007 at ¶¶ 66-69.

Claim 13	**Bodorin**
The method of claims 1, further comprising	Ex. 1003 at 1:44-61 (emphasis added): As shown in FIG. 1A, a malware 102 is directed over a network 106 to the computer 110, as indicated by arrow 108. It will be

blocking access to the software when the software is classified as malware or unwanted software.	appreciated that the malware 102 may be directed to the computer 110 as a result of a request initiated by the computer, or directed to the computer from another network device. However, as mentioned above, before the known malware 102 reaches the computer 110, anti-virus software 104 installed on the computer intercepts the malware and examines it. As is known in the art, currently, anti-virus software scans the incoming data as a file, searching for identifiable patterns, also referred to as signatures, associated with known-malware. If a malware signature is located in the file, the anti-virus software 104 takes appropriate action, such as deleting the known malware/infected file, or removing the malware from an infected file, sometimes referred to as cleaning the file. ***In this manner, anti-virus software 104 is able to prevent the known malware 102 from infecting the computer 110, as indicated by the arrow 112***. *See also* Ex. 1007 at ¶¶ 66-69.

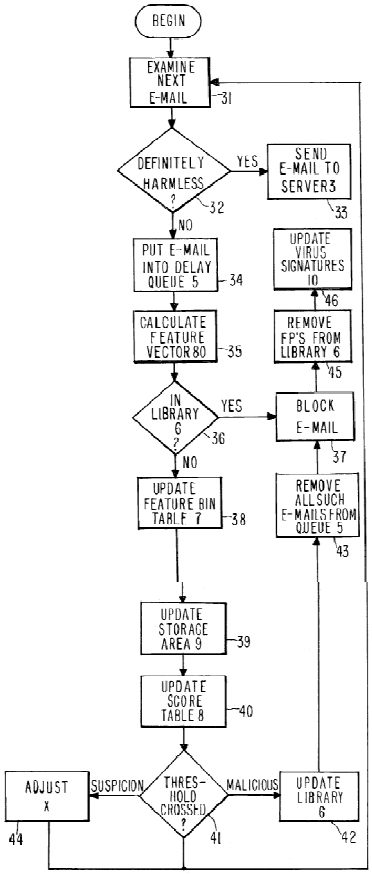
C. Claims 1, 2, 10, and 13 Are Anticipated by Kissel

1. Statement of Non-Redundancy

The grounds raised in the below sections are meaningfully distinct from each other and rely upon substantively different combinations of the cited prior art. The grounds detailed in Sections VII.C-D rely upon Kissel as the primary reference. The grounds detailed in Section VII.A-B, and E rely upon Bodorin as the primary reference. Whereas Kissel discloses a system for detecting the presence of malicious computer code in emails, Bodorin discloses a malware detection system that determines whether an executable code module is malware according to behaviors exhibited by the code module while executing. As Patent Owner may attempt to distinguish the challenged claims by arguing the '344 taught away from

executing suspected malware software to observe behaviors, Kissel should be included for trial. As Patent Owner may attempt to distinguish the challenged claims based on whether a tokenized representation of a script program within an email as disclosed in Kissel constitute APIs as recited in the claims, Bodorin should be included for trial.

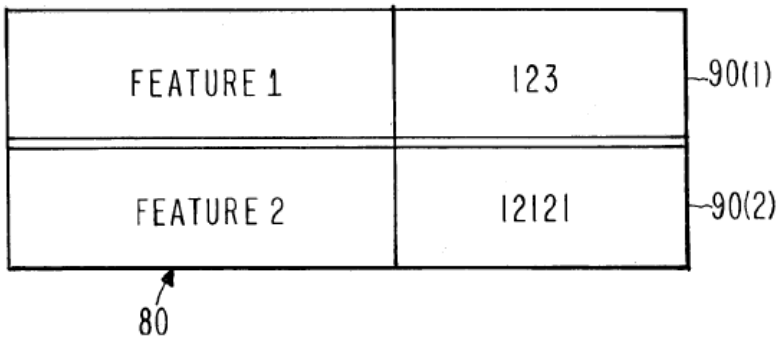
Kissel discloses a system for detecting the presence of malicious computer code in emails. Ex. 1004 at 1:44-46. The operation of the system is generally illustrated by Figs. 3A and 3B:



Ex. 1004 at Figs. 3A and 3B.

As shown in the figure, if the email is not deemed to be definitely harmless, it is put into a delay queue for further processing. Ex. 1004 at 5:1-19. At step 35, the system disclosed by Kissel extracts a feature vector from the email being examined. Ex. 1004 at 5:20-24. Features that may be extracted include contents of the subject line of the email, a hash of the email attachment, name of the email attachment, etc. Ex. 1004 at 3:35-46. Notably, emails may contain software such as JavaScript programs and the features extracted may also include functionality of that software, such as a tokenized representation of a script program within an email, which may include JavaScript API calls. Ex. 1004 at 3:45-4:4. Keywords, for example "var" and "if," are searched for within this set of APIs and strings. Ex. 1004 at 3:45-4:4. The extract feature vector is shown in Fig. 4.

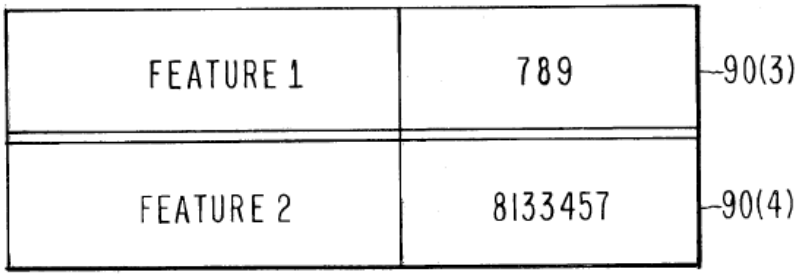
FIG. 4



Ex. 1004 at Figs. 4.

This feature vector is then examined against the blocked feature instance library in step 36. Exemplary content of the library is shown in Fig. 5.

FIG.5



Ex. 1004 at Figs. 5.

If the extracted features match those in the library, then the email is blocked. Ex. 1004 at 5:47-60. If however, the extracted features do not match those in the library, then additional processing is done at steps 38-41 to determine if the feature vector under examination crosses a certain threshold. Ex. 1004 at 6:7-8:62. If it exceeds the threshold, then the email is deemed to be malicious and the library is updated at step 42. *Id.* Notably, the threshold levels are calibrated by testing for false-positives. Ex. 1004 at 8:58-62.

Each and every limitation of the Claims 1, 2, 10, and 13 of the '344 Patent are disclosed by Kissel, in the same arrangement as in the '344 Patent, and are therefore anticipated.

Claim 1	**Kissel**
A method for classifying software, said method comprising;	Ex. 1004 at Abstract: Methods, apparati, and computer-readable media for detecting the presence of malicious computer code in a plurality of e-mails. *See also* Ex. 1007 at ¶ 77.

[a] providing a library of gene information including a number of classifications based on groupings of genes;	Ex. 1004 at 3:23-30 (emphasis added): Coupled to gateway 2 is a delay queue 5, a storage area for temporarily holding e-mails while the method steps of the present invention are executed. As used herein, "coupled" means any type of direct or indirect coupling, or direct or indirect connection. ***Blocked feature instance library 6, also coupled to gateway 2, is a storage area containing exemplars of feature instances (values) that have been deemed to be indicative of malicious code***. Ex. 1004 at 5:32-47: At step 36, gateway 2 inquires whether a feature entry 90 calculated in step 35 corresponds to a blocked feature instance deemed to be indicative of malicious code, as stored in library 6. In the example illustrated in FIG. 5, there are only two feature instances (90(3) and 90(4)) that have been determined previously to be indicative of malicious code. The only instance for feature 1 that has been determined to be malicious is 789. This may represent, for example, the hash of "with love from Pamela" in the subject line of the e-mail. This does not match the instance 90(1) of feature 1 extracted in step 35 and illustrated in FIG. 4. Similarly, the only instance of feature 2 previously found to have been associated with malicious code is 8133457, as illustrated in FIG. 5. This could be the hash of an e-mail attachment containing a worm. This does not match the instance 90(2) of feature 2 illustrated in FIG. 4 either. *See also* Ex. 1007 at ¶ 78.
[b] identifying at least one functional block and at least one property of the software;	Ex. 1004 at 3:34-64: Examples of e-mail features that may be tracked by the present invention include: Contents of the subject line of the e-mail. A hash of each e-mail attachment (or, not as commonly due to the normally large size of an e-mail attachment, the entire attachment itself). The name of each e-mail attachment. The size of each e-mail attachment. A hash of the e-mail body. Parsed script within the e-mail body or within an e-mail attachment. A hash of parsed script within the e-mail body or within an e-mail attachment. To illustrate the concept of "parsed script", let us examine the following example of what could be included within an e-

	mail body:
	
	`<HTML>`
	`<SCRIPT language = "JavaScript">`
	`<!--`
	`var userAgent=navigator.userAgent;`
	`. . .`
	`//-->`
	`</SCRIPT>`
	
	We look only between the <SCRIPT> tags. What is there is script (code), in this case script in the JavaScript language. We then parse the script using a parser, much like an interpreter would, to generate a tokenized representation of the script.
	
	See also Ex. 1007 at ¶ 79.

[c] identifying one or more genes each describing one or more of the at least one functional block and the at least one property of the software as a sequence of APIs and strings	Ex. 1004 at 3:34-4:4: Examples of e-mail features that may be tracked by the present invention include: Contents of the subject line of the e-mail. A hash of each e-mail attachment (or, not as commonly due to the normally large size of an e-mail attachment, the entire attachment itself). The name of each e-mail attachment. The size of each e-mail attachment. A hash of the e-mail body. Parsed script within the e-mail body or within an e-mail attachment. A hash of parsed script within the e-mail body or within an e-mail attachment. To illustrate the concept of "parsed script", let us examine the following example of what could be included within an e-mail body:
	
	`<HTML>`
	`<SCRIPT language = "JavaScript">`
	`<!--`
	`var userAgent=navigator.userAgent;`
	`. . .`
	`//-->`
	`</SCRIPT>`
	
	We look only between the <SCRIPT> tags. What is there is script (code), in this case script in the JavaScript language. We then parse the script using a parser, much like an interpreter would, to generate a tokenized representation of

	the script. We look for keywords. For example, after "var" (variable), we don't care what the variable is, so the parsed representation might be var* var* if* where * denotes anything. Ex. 1004 at 5:20-31: At step 35, gateway 2 calculates a feature vector 80 for the e-mail, based upon the preselected set of features. As an example, assume that there are two features to be examined, the two features that were described above in conjunction with FIG. 2. Then feature vector 80 thus normally contains two entries 90, one for each feature. FIG. 4 illustrates that as an example, feature 1 is calculated to be 123; this corresponds to instance 1 of feature 1 on FIG. 2. Feature 2 is calculated to be 12121, which corresponds to instance 2 of feature Z on FIG. 2. *See also* Ex. 1007 at ¶ 80.
[d] matching the one or more genes against one or more of the number of classifications using a processor;	Ex. 1004 at 5:32-35: At step 36, gateway 2 inquires whether a feature entry 90 calculated in step 35 corresponds to a blocked feature instance deemed to be indicative of malicious code, as stored in library 6. *See also* Ex. 1007 at ¶ 81.
[e] classifying the software based on the matching to provide a classification for the software; and	Ex. 1004 at 5:48-52: If, however, one of the feature instances had matched a blocked feature instance contained in library 6, the method would have proceeded to step 37, where the e-mail, now deemed to contain malicious code, is blocked. *See also* Ex. 1007 at ¶ 81.
[f] notifying a user of the classification of the software.	Ex. 1004 at 5:52-54: As used herein, "blocked" can mean at least one of the following: The system administrator is alerted to the problem. *See also* Ex. 1007 at ¶ 82.

Claim 2	Kissel
The method of claim 1, wherein the classification for the software includes malware.	Ex. 1004 at 5:48-52 (emphasis added): If, however, one of the feature instances had matched a blocked feature instance contained in library 6, the method would have proceeded to step 37, where the e-mail, ***now deemed to contain malicious code***, is blocked. Ex. 1004 at 1:13-25: As used herein, "malicious computer code" is any computer code that enters a computer without an authorized user's knowledge and/or without an authorized user's consent. The malicious code typically performs malicious actions after it arrives on the user's computer. Malicious computer code that propagates from one computer to another over a network, e.g., via e-mail, is often referred to as a "worm" or "spam". A worm is self-propagating, e.g., after it enters a user's computer, it may spread to other computers by attaching itself to e-mails that are sent to addresses found in the first computer's address book. Spam, on the other hand, is unwanted e-mail received by a computer that does not self-propagate. *See also* Ex. 1007 at ¶ 82.

Claim 10	Kissel
The method of claim 1, further comprising removing the software when the software is classified as malware or unwanted software.	Ex. 1004 at 5:52-55 (emphasis added): As used herein, "blocked" can mean at least one of the following: The system administrator is alerted to the problem. ***The e-mail is deleted from delay queue 5***. *See also* Ex. 1007 at ¶ 82.

Claim 13	Kissel
The method of claims 1, further comprising blocking access to the software when the software is classified as malware or unwanted software.	Ex. 1004 at 5:52-58 (emphasis added): As used herein, "blocked" can mean at least one of the following: The system administrator is alerted to the problem. The e-mail is deleted from delay queue 5. ***The e-mail is quarantined and sent to an antivirus research center such as Symantec Antivirus Research Center in Santa Monica, Calif.*** *See also* Ex. 1007 at ¶ 82.

D. Claims 16 and 17 Are Rendered Obvious in Light of Kissel Further in View of Apap

1. Apap

Apap discloses a method for detecting intrusions in the operation of a computer system. Ex. 1006 at Abstract. The system disclosed by Apap first generates a model for normal behavior by a computer system. Ex. 1006 at 5:10-20. An algorithm for detecting anomalous behavior is then employed, and the algorithm is trained and tested by comparing the detected anomalous behavior against normal behavior to generate statistics related to detection and false positives. Ex. 1006 at 15:44-16:20.

2. It Would Have Been Obvious to Combine Kissel and Apap

As explained above, Kissel discloses that when the extracted features do not match those in the library, then additional processing is done at steps 38-41 to determine if the feature vector under examination crosses a certain threshold. Ex. 1004 at 6:7-8:62. If it exceeds the threshold, then the email is deemed to be

malicious and the library is updated at step 42. *Id.* The threshold levels are calibrated by testing for false-positives. Ex. 1004 at 8:58-62. To the extent Kissel does not disclose testing for false-positives against one or more reference files, Apap discloses testing its algorithm for detecting anomalous behavior by referencing the detected anomalies against recorded normal behavior. Accordingly, Kissel combined with Apap discloses every limitation of the Claims 16 and 17 of the '344 patent.

It would have been obvious to a person of ordinary skill in the art to combine Kissel and Apap so that suspected malware is tested for false-positives against one or more reference files. As explained by Dr. Nielson, testing, for example, suspected malware, for false-positives by using a reference that indicates whether the suspected malware is indeed malware was a well understood method known in the art to test for false-positives. Ex. 1007 at ¶¶ 86-88. The fact that this method was well known is demonstrated not only by Apap itself, but also by U.S. Patent No. 8,713,686 to Kane, filed on January 25, 2006, which discloses a system for virus detection that reduces false-positives by testing the suspected viruses against a database of known non-viruses. Ex. 1010 at 4:38-43. Thus, there was explicit teaching, suggestion, and motivation in the art to combine Kissel and Apap. Ex. 1007 at ¶¶ 86-88. Combining Kissel and Apap is also a combination that unites old elements with no change in their respective functions, would not

present technical challenges, and would not negatively impact the functionality of Kissel's malware detection system. Ex. 1007 at ¶ 89. As explained above, Kissel already discloses checking suspected malware for false-positives. Kissel does not explicitly disclose using one or more reference files to performing the operation of testing for false-positives, but because using reference files was a commonly understood way to perform such testing, a combination of Kissel and Apap would unite old elements with no change in their respective functions. Ex. 1007 at ¶ 89.

The following claim charts demonstrate, on a limitation-by-limitation basis, how Claims 16 and 17 are rendered obvious by Kissel in combination with Apap.

Claim 16	**Kissel and Apap**
A method for generating software classifications for use in classifying software, said method comprising:	Kissel discloses a method to update its blocked feature instance library for use the classifying malicious code. *See* Ex. 1004 at 6:7-8:62. *See also* Ex. 1007 at ¶ 83.
[a] providing a library of gene information including a number of classifications based on groupings of genes;	Ex. 1004 at 3:23-30 and 5:32-47. *See also* Ex. 1007 at ¶ 78.
[b] identifying one or more genes each describing a	Ex. 1004 at 3:34-4:4: Examples of e-mail features that may be tracked by the present invention include: Contents of the subject line of the e-mail. A hash of each e-mail attachment

functionality or a property of the software as a sequence of APIs and strings;	(or, not as commonly due to the normally large size of an e-mail attachment, the entire attachment itself). The name of each e-mail attachment. The size of each e-mail attachment. A hash of the e-mail body. Parsed script within the e-mail body or within an e-mail attachment. A hash of parsed script within the e-mail body or within an e-mail attachment. To illustrate the concept of "parsed script", let us examine the following example of what could be included within an e-mail body:
	
	`<HTML>`
	`<SCRIPT language = "JavaScript">`
	`<!--`
	`var userAgent=navigator.userAgent;`
	`. . .`
	`//-->`
	`</SCRIPT>`
	
	We look only between the <SCRIPT> tags. What is there is script (code), in this case script in the JavaScript language. We then parse the script using a parser, much like an interpreter would, to generate a tokenized representation of the script. We look for keywords. For example, after "var" (variable), we don't care what the variable is, so the parsed representation might be
	var*
	var*
	if* where * denotes anything.
	
	Ex. 1004 at 5:20-31: At step 35, gateway 2 calculates a feature vector 80 for the e-mail, based upon the preselected set of features. As an example, assume that there are two features to be examined, the two features that were described above in conjunction with FIG. 2. Then feature vector 80 thus normally contains two entries 90, one for each feature. FIG. 4 illustrates that as an example, feature 1 is calculated to be 123; this corresponds to instance 1 of feature 1 on FIG. 2. Feature 2 is calculated to be 12121, which corresponds to instance 2 of feature Z on FIG. 2.
	
	See also Ex. 1007 at ¶ 80.

[c] combining a plurality of genes that describe the software, thereby providing a set of genes;	Ex. 1004 at 5:20-31 (emphasis added): At step 35, gateway 2 calculates ***a feature vector 80 for the e-mail, based upon the preselected set of features***. As an example, assume that there are two features to be examined, the two features that were described above in conjunction with FIG. 2. Then feature vector 80 thus normally contains two entries 90, one for each feature. FIG. 4 illustrates that as an example, feature 1 is calculated to be 123; this corresponds to instance 1 of feature 1 on FIG. 2. Feature 2 is calculated to be 12121, which corresponds to instance 2 of feature Z on FIG. 2. *See also* Ex. 1007 at ¶ 80.
[d] testing the set of genes for false-positives against one or more reference files using a processor;	Ex. 1004 at 6:7-28: If none of the feature instances extracted in 35 correspond to an entry in library 6, step 38 is entered, during which gateway 2 updates feature bin table 7 with one entry for each feature that has been extracted. As stated previously, an entry consists of a message ID and a time index. At step 39, gateway 2 checks the timestamps of all of the entries within feature bin table 7. If any entry has a timestamp outside a preselected window of time Q (which is the sample time used in calculating score S), that entry is removed from feature bin table 7 and placed into recently removed entry storage area 9, which stores those entries that have been removed from table 7 since the last time the method steps of FIG. 3 were executed. At step 40, all scores S in score table 8 are updated. At step 41, gateway 2 determines whether any of the recently updated scores exceed a preselected malicious threshold for that score, indicative of malicious code. If this malicious threshold has been crossed, library 6 at step 42 is updated with an entry corresponding to the offending feature instance. Then, at step 43, all e-mails in queue 5 containing this feature instance are deleted, and the method reverts to step 37. Ex. 1004 at 8:47-62 (emphasis added): Then, an offline analysis is performed on these logs to investigate an optimum combination of values for w_R, w_P, h_R, and h_P, as well as the time window size Q. Since this is a multi-parameter

	optimization problem that is further complicated by different spreading characteristics of different (known and unknown) e-mail worms, a multi-parameter non-linear best fit (in the least mean squares sense) optimization algorithm (such as a neural network) can advantageously be used. ***However, visual investigation of the worm spreading characteristics, along with manual "tweaking" of the scoring parameters, may result in a satisfactory set of parameters. This manual setting of the scoring parameters also gives system administrators the option to customize the behavior of the invention, in particular, the false positive threshold appropriate for their computing environment***. Ex. 1006 at 15:44-51 (emphasis added): ***The first exemplary anomaly detection algorithm discussed above in equations (1)-(3) were trained over the normal data***. Each record in the testing set was evaluated against this training data***. The results were evaluated by computing two statistics: the detection rate and the false positive rate.*** The performance of the system was evaluated by measuring detection performance over processes labeled as either normal or malicious. *See also* Ex. 1007 at ¶¶ 83-90.
[e] defining the software classification based on the set of genes; and	Ex. 1004 at 6:20-28 (emphasis added): At step 40, all scores S in score table 8 are updated. At step 41, ***gateway 2 determines whether any of the recently updated scores exceed a preselected malicious threshold for that score, indicative of malicious code***. If this malicious threshold has been crossed, library 6 at step 42 is updated with an entry corresponding to the offending feature instance. Then, at step 43, all e-mails in queue 5 containing this feature instance are deleted, and the method reverts to step 37. *See also* Ex. 1007 at ¶ 83.
[f] storing the set of genes and the software classification in the	Ex. 1004 at 6:20-28 (emphasis added): At step 40, all scores S in score table 8 are updated. At step 41, gateway 2 determines whether any of the recently updated scores exceed a preselected malicious threshold for that score,

library.	indicative of malicious code. ***If this malicious threshold has been crossed, library 6 at step 42 is updated with an entry corresponding to the offending feature instance***. Then, at step 43, all e-mails in queue 5 containing this feature instance are deleted, and the method reverts to step 37. *See also* Ex. 1007 at ¶ 83.

Claim 17	**Kissel and Apap**
The method of claim 16, wherein the software classification includes malware.	Ex. 1004 at 1:13-25 and 5:48-52. *See also* Ex. 1007 at ¶ 82.

E. Claims 16 and 17 Are Rendered Obvious in Light of Bodorin Further in View of Kissel and Apap.

As explained above, Bodorin discloses a malware detection system that generates behavior signatures from programs, which corresponds to the groupings or sets of genes in the claims of the '344 Patent, and compares it against behavior signatures of known malware in the malware behavior signature store, which corresponds to the "library" in the claims of the '344 Patent. Bodorin does not disclose updating the malware behavior signature store by testing the generated behavior signature for false positives and storing the tested behavior signature, which is claimed in Claim 16. However, as explained in detail in Section VII.D, Kissel combined with Apap disclose every limitation of Claim 16.

It would have been obvious to a person of ordinary skill in the art to combine Bodorin with Kissel and Apap. There was explicit teaching, suggestion, and motivation in the prior art references themselves to combine. Claim 16 discloses a method for testing a generated set of genes for false-positives and storing the set of genes. Kissel discloses this method as a way of updating the library. Ex. 1004 at 6:7-8:62. Although not explicitly disclosed in Kissel, Apap discloses testing for false-positives using one or more reference files. Ex. 1006 at 15:44-51. Although Bodorin does not explicitly disclose a method for updating its malware behavior signature store, Bodorin nevertheless explicitly states that it is desirable to update the store. Specifically, Bodorin states "[i]t is anticipated that the malware behavior signature store 208 will be periodically updated with behavior signatures of known malware. The malware behavior signature store 208 should be especially updated as the more rare, 'new' malware is discovered." Ex. 1003 at 6:1-5. Accordingly, one of ordinary skill in the art reading this passage in Bodorin would be motivated to combine the disclosures of Bodorin with a method for updating the malware behavior signature store, which is exactly what is disclosed in Kissel and Apap.

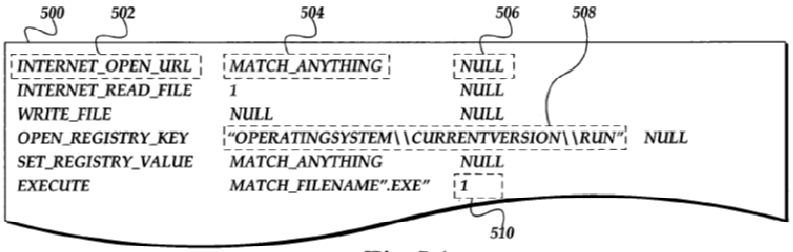
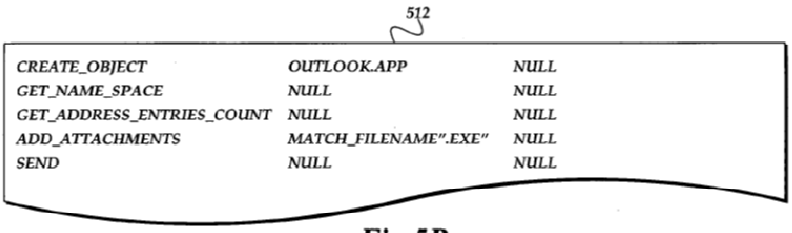
Furthermore, there was explicit teaching, suggestion, and motivation generally in the art to combine. As explained by Dr. Nielson, new malware appear regularly and frequently. Ex. 1007 at ¶ 93. A major challenge for makers of anti-

virus software is to keep its anti-virus software up to date so that it can recognize new malware before the new malware can infect users' computers. *Id.* Thus, one of ordinary skill in the art, in view of a system disclosed in Bodorin that generates behavior signature from programs and compares it against behavior signatures of known malware in the malware behavior signature store, would be motivated to keep the stored updated so that the system can detect the latest malware. Accordingly, one of ordinary skill in the art would be motivated to combine Bodorin with Kissel and Apap. *See* Ex. 1007 at ¶¶ 91-94.

The following claim charts demonstrate, on a limitation-by-limitation basis, how Claims 16 and 17 are rendered obvious by Bodorin further in view of Kissel and Apap.

Claim 16	**Bodorin further in view of Kissel and Apap**
A method for generating software classifications for use in classifying software, said method comprising:	Bodorin discloses that the malware behavior signature store should be updated. Ex. 1003 at 6:1-5. Kissel discloses a method to update its blocked feature instance library for use the classifying malicious code. *See* Ex. 1004 at 6:7-8:62. *See also* Ex. 1007 at ¶ 83.
[a] providing a library of gene information including a number of classifications based on groupings	Ex. 1003 at 5:27-43. *See also* Ex. 1007 at ¶¶ 59-61.

of genes;	
[b] identifying one or more genes each describing a functionality or a property of the software as a sequence of APIs and strings;	Ex. 1003 at 4:51-5:20: As the code module 212 executes within the dynamic behavior evaluation module 204, the dynamic behavior evaluation module records "interesting" behaviors exhibited by the code module. Interesting behaviors are those which a user or implementer of the malware detection system 200 has identified as interesting, potentially associated with malware, and are used to compare the behaviors of the code module 212 against known malware behaviors. Table A includes an exemplary, representative list of "interesting" behaviors, including parameters that are also considered interesting, that are recorded by a dynamic behavior evaluation module 204. It should be understood that these listed "interesting" behaviors are exemplary only, and should not be construed as limiting upon the present invention. Those skilled in the art will recognize that each individual type of code module may include a unique set of interesting behaviors. Additionally, as new features are added to each environment, or as computer systems change, additional behaviors may be added as "interesting," while others may be removed. TABLE A RegisterServiceProcess ( ) Sleep ( ) WinExec (application_name) InternetOpenUrlA (URL_name) InternetOpenA (URL_name) IntenetClosetHandle ( ) CreateFileA (file_name) ReadFile ( ) WriteFile ( ) Close Handle ( ) RegOpenKeyA (key_name) RegSetValueA (subkey_name) RegSetValuExA (subkey_name) RegCloseKey ( ) UrlDownloadToFileA (url_name, file_name) ExitProcess ( ) GetCommandLine ( ) CreateProcessA (proces_name, parameters_list) GlobalAlloc ( ) GetTickCount ( ) CopyFileA (new_name, existing_name) GetWindowsDirectoryA( ) GetSystemDirectoryA ( ) GetModuleFileNameA ( ) LoadLibraryA (library_name) GetProcAddress (procedure_name) FindFirstFileA (file_specificator) FindNextFileA ( ) FindClose ( ) Ex. 1003 at Claim 1 (emphasis added): . . . at least one dynamic behavior evaluation module, wherein each dynamic

	behavior evaluation module provides a virtual environment for executing a code module of a particular type, and wherein ***each dynamic behavior evaluation module records interesting API function calls that the code module makes as it is executed***, wherein the interesting API function calls are specified by a user and comprise a portion of all API function calls that the code module makes, wherein only the interesting API function calls, but not all API function calls, that the code module makes during execution in the dynamic behavior evaluation module are recorded into a behavior signature corresponding to the code module; *See also* Ex. 1007 at ¶ 62.
[c] combining a plurality of genes that describe the software, thereby providing a set of genes;	Ex. 1003 at 7:16-22: FIGS. 5A and 5B are block diagrams illustrating exemplary behavior signatures of hypothetical code modules, such as the behavior signature 210 described above. Each row of items in the exemplary behavior signatures, including behavior signature 500 and behavior signature 512, represents a behavior that was exhibited by a code module and recorded by the dynamic behavior evaluation module 204.  500 502 504 506 508 INTERNET_OPEN_URL MATCH_ANYTHING NULL INTERNET_READ_FILE 1 NULL WRITE_FILE NULL NULL OPEN_REGISTRY_KEY "OPERATINGSYSTEM\\CURRENTVERSION\\RUN" NULL SET_REGISTRY_VALUE MATCH_ANYTHING NULL EXECUTE MATCH_FILENAME".EXE" 1 510 *Fig.5A.*  512 CREATE_OBJECT OUTLOOK.APP NULL GET_NAME_SPACE NULL NULL GET_ADDRESS_ENTRIES_COUNT NULL NULL ADD_ATTACHMENTS MATCH_FILENAME".EXE" NULL SEND NULL NULL *Fig.5B.* *See also* Ex. 1007 at ¶ 63.
[d] testing the set	Ex. 1004 at 6:7-28: If none of the feature instances extracted

of genes for false-positives against one or more reference files using a processor;	in 35 correspond to an entry in library 6, step 38 is entered, during which gateway 2 updates feature bin table 7 with one entry for each feature that has been extracted. As stated previously, an entry consists of a message ID and a time index. At step 39, gateway 2 checks the timestamps of all of the entries within feature bin table 7. If any entry has a timestamp outside a preselected window of time Q (which is the sample time used in calculating score S), that entry is removed from feature bin table 7 and placed into recently removed entry storage area 9, which stores those entries that have been removed from table 7 since the last time the method steps of FIG. 3 were executed. At step 40, all scores S in score table 8 are updated. At step 41, gateway 2 determines whether any of the recently updated scores exceed a preselected malicious threshold for that score, indicative of malicious code. If this malicious threshold has been crossed, library 6 at step 42 is updated with an entry corresponding to the offending feature instance. Then, at step 43, all e-mails in queue 5 containing this feature instance are deleted, and the method reverts to step 37. Ex. 1004 at 8:47-62 (emphasis added): Then, an offline analysis is performed on these logs to investigate an optimum combination of values for w_R, w_P, h_R, and h_P, as well as the time window size Q. Since this is a multi-parameter optimization problem that is further complicated by different spreading characteristics of different (known and unknown) e-mail worms, a multi-parameter non-linear best fit (in the least mean squares sense) optimization algorithm (such as a neural network) can advantageously be used. ***However, visual investigation of the worm spreading characteristics, along with manual "tweaking" of the scoring parameters, may result in a satisfactory set of parameters. This manual setting of the scoring parameters also gives system administrators the option to customize the behavior of the invention, in particular, the false positive threshold appropriate for their computing environment***. Ex. 1006 at 15:44-51 (emphasis added): ***The first exemplary anomaly detection algorithm discussed above in equations***

	(1)-(3) were trained over the normal data. Each record in the testing set was evaluated against this training data***. The results were evaluated by computing two statistics: the detection rate and the false positive rate.*** The performance of the system was evaluated by measuring detection performance over processes labeled as either normal or malicious. *See also* Ex. 1007 at ¶¶ 83-89.
[e] defining the software classification based on the set of genes; and	Ex. 1004 at 6:20-28 (emphasis added): At step 40, all scores S in score table 8 are updated. At step 41, ***gateway 2 determines whether any of the recently updated scores exceed a preselected malicious threshold for that score, indicative of malicious code***. If this malicious threshold has been crossed, library 6 at step 42 is updated with an entry corresponding to the offending feature instance. Then, at step 43, all e-mails in queue 5 containing this feature instance are deleted, and the method reverts to step 37. *See also* Ex. 1007 at ¶ 83.
[f] storing the set of genes and the software classification in the library.	Ex. 1004 at 6:20-28 (emphasis added): At step 40, all scores S in score table 8 are updated. At step 41, gateway 2 determines whether any of the recently updated scores exceed a preselected malicious threshold for that score, indicative of malicious code. ***If this malicious threshold has been crossed, library 6 at step 42 is updated with an entry corresponding to the offending feature instance***. Then, at step 43, all e-mails in queue 5 containing this feature instance are deleted, and the method reverts to step 37. *See also* Ex. 1007 at ¶ 83.

Claim 17	**Bodorin further in view of Kissel and Apap**
The method of claim 16, wherein the software	Ex. 1003 at 5:44-49. *See also* Ex. 1007 at ¶ 64.

classification includes malware.	

F. Claims 1, 2, 10, and 13 Are Anticipated by Chen

1. Statement of Non-Redundancy

The grounds raised in the below sections are meaningfully distinct from each other and rely upon substantively different combinations of the cited prior art. The grounds detailed in Sections VII.C-D rely upon Kissel as the primary reference. The grounds detailed in Section VII.F rely upon Chen as the primary reference. Whereas Kissel discloses a system for detecting the presence of malicious computer code in emails, Chen discloses a system to detect and remove viruses present in macros by scanning the macros. As Patent Owner may attempt to distinguish the challenged claims based on whether scanning macros meets the claims, Kissel should be included for trial. As Patent Owner may attempt to distinguish the challenged claims based on whether a tokenized representation of a script program within an email as disclosed in Kissel constitute APIs as recited in the claims, Chen should be included for trial.

The grounds detailed in Section VII.A-B and E rely upon Bodorin as the primary reference. As Patent Owner may attempt to distinguish the challenged claims based on whether scanning macros meets the claims, Bodorin should be included for trial. As Patent Owner may attempt to distinguish the challenged

claims by arguing the ‘344 taught away from executing suspected malware software to observe behaviors, Chen should be included for trial.

Chen discloses a system to detect and remove viruses present in macros. Ex. 1005 at Abstract. Generally, macros are an executable series of instructions including menu selections, keystrokes and/or commands that are recorded and assigned a name or a key. Ex. 1005 at 1:39-41. The virus detection operation as disclosed in Chen is illustrated by Fig. 6.

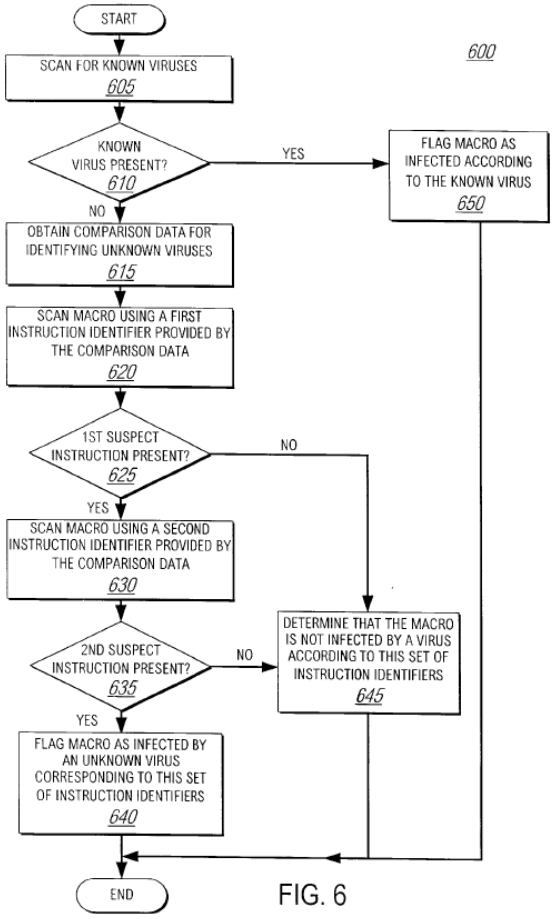


FIG. 6

Ex. 1005 at Fig. 6.

In the first step 605, a decoded macro is scanned for known viruses. If known viruses are found, then the macro is flagged. Ex. 1005 at 13:7-19. If no known viruses are found, then the system scans for unknown viruses. The virus scanning module of the system obtains a set of instruction identifiers from the virus information module. Ex. 1005 at 13:20-33. The virus information module is a library that includes sets of instruction identifiers which are used to identify combinations of suspect instructions in the macro. Ex. 1005 at 8:58-54. An exemplary set of instruction identifiers is shown in Fig. 9. As shown in Fig. 9, these sets of instruction identifiers include instructions or API calls such as MacroCopy or FileSaveAs, and also strings in hexadecimal form, such as 73 CB 00 0C 6C 01 00. Ex. 1005 at 13:64-14:51. If a macro containing a virus is identified, then the macro is treated by removing the virus. Ex. 1005 at 16:1-50.

Each and every limitation of the Claims 1, 2, 10, and 13 of the '344 Patent are disclosed by Chen, in the same arrangement as in the '344 Patent, and are therefore anticipated.

Claim 1	**Chen**
A method for classifying software, said method comprising;	Ex. 1005 at Abstract: The detection and removal of viruses from macros is disclosed. A macro virus detection module includes a macro locating and decoding module, a macro virus scanning module, a macro treating module, a file treating module, and a virus information module. The macro locating and decoding module determines whether a targeted file includes a macro, and, where a macro is found, locates and decodes it to produce a decoded macro. The macro virus

	scanning module accesses the decoded macro and scans it to determine whether it includes any viruses. Unknown macro viruses are detected by the macro virus scanning module by obtaining comparison data that includes sets of instruction identifiers from the virus information module and determining whether the decoded macro includes a combination of suspect instructions which correspond to instruction identifiers. The macro treating module locates suspect instructions in the decoded macro using the comparison data and removes the suspect instructions to produce a treated macro. The file correcting module accesses a targeted file with an infected macro and replaces the infected macro with the treated macro produced by the macro treating module. *See also* Ex. 1007 at ¶ 99.

[a] providing a library of gene information including a number of classifications based on groupings of genes;	Ex. 1005 at 8:58-9:4: To identify suspect instruction combinations, the macro virus scanning module 304 accesses comparison data from the virus information module 308. The comparison data includes sets of instruction identifiers which are used to identify combinations of suspect instructions in the decoded macro. An exemplary set of instruction identifiers includes first and second suspect instruction identifiers. The instruction identifiers are binary strings and the decoded macros are scanned to determine whether they include the binary strings and, accordingly, the suspect instructions. If it is determined that the macro includes the combination of suspect instructions defined and identified by the set of instruction identifiers, then it is determined that the macro is infected by an unknown virus corresponding to that set of data. Ex. 1005 at 14:65-15:13: Referring now to FIG. 9, a data table including exemplary sets of instruction identifiers which are stored in the virus information module 308 is shown. The exemplary data table 900 includes rows 902 which correspond to several different sets of instruction identifiers. Columns identifying the sets of instruction identifiers 903, their instruction identifier numbers 904 and text and corresponding hexadecimal representations 905 of the binary code for the instruction identifiers are included. Preferably, two instruction identifiers are included in each set of instruction identifiers, but additional instruction identifiers can be included in a set. Additionally, a positive macro virus determination can be made based upon detection of two out of three instruction identifiers or some other subset of identifiers. The data table 900 is exemplary only. The comparison data may be variously stored in the virus information module 308. Ex. 1005 at Fig. 9. *See also* Ex. 1007 at ¶ 100.
[b] identifying at least one functional block and at least	Ex. 1005 at 13:20-33: If in step 610 it is determined that known viruses were not detected in the scanning step 605, the macro virus scanning module 304 scans for unknown

one property of the software;	viruses in the decoded macros. In step 615, the macro virus scanning module 304 obtains a set of instruction identifiers for detecting unknown viruses. The macro scanning module 304 detects instructions which are likely to be used by macro viruses. These instructions are also referred to as suspect instructions. Certain combinations of suspect instructions are very likely to be used macro viruses. By looking for combinations of suspect instructions, false positive virus detection is avoided since a macro with two (or more) different suspect instructions is very likely to be infected. *See also* Ex. 1007 at ¶ 101.
[c] identifying one or more genes each describing one or more of the at least one functional block and the at least one property of the software as a sequence of APIs and strings	Ex. 1005 at 15:13-31: Referring again to the flow chart of FIG. 6, once a set of instruction identifiers is obtained by the macro virus scanning module 304 in step 615, the decoded macro is scanned to determine whether it includes a combination of suspect instructions as identified by the instruction identifiers. In step 620, the decoded macro is scanned using a first instruction identifier. For example, the decoded macro may be scanned 620 to determine whether a string 73 CB 00 0C 6C 01 00 which corresponds to the first instruction identifier in the first set of exemplary instruction identifiers 900 is present. The scanning in step 620 may be undertaken by a state machine which scans the decoded macro and determines whether the string is present. In step 625, it is determined whether the first suspect instruction identifier is present in the decoded macro. If it is determined 625 that the no instruction corresponding to the first instruction identifier is present, then in step 645 it is determined that the macro is not infected according to the set of instruction identifiers and the method of macro virus scanning 600 ends. *See also* Ex. 1007 at ¶ 102.
[d] matching the one or more genes against one or more of the number of	Ex. 1005 at 15:13-31: Referring again to the flow chart of FIG. 6, once a set of instruction identifiers is obtained by the macro virus scanning module 304 in step 615, the decoded macro is scanned to determine whether it includes a combination of suspect instructions as identified by the

classifications using a processor;	instruction identifiers. In step 620, the decoded macro is scanned using a first instruction identifier. For example, the decoded macro may be scanned 620 to determine whether a string 73 CB 00 0C 6C 01 00 which corresponds to the first instruction identifier in the first set of exemplary instruction identifiers 900 is present. The scanning in step 620 may be undertaken by a state machine which scans the decoded macro and determines whether the string is present. In step 625, it is determined whether the first suspect instruction identifier is present in the decoded macro. If it is determined 625 that the no instruction corresponding to the first instruction identifier is present, then in step 645 it is determined that the macro is not infected according to the set of instruction identifiers and the method of macro virus scanning 600 ends. *See also* Ex. 1007 at ¶ 102.
[e] classifying the software based on the matching to provide a classification for the software; and	Ex. 1005 at 15:43-54: If it is determined in step 625 that the first instruction identifier is present, then the decoded macro is scanned in step 630 to determine whether the second instruction identifier is present. If in step 635 it is determined that the macro includes the second suspect instruction identifier, then in step 640 the decoded macro is flagged as infected by an unknown virus corresponding to the set of instruction identifiers. Information associating the decoded macro to the set of instruction identifiers that resulted in a positive unknown virus detection is stored in the data buffer 312 so that other modules such as the macro treating module 306 can treat the infected macro accordingly. *See also* Ex. 1007 at ¶ 102.
[f] notifying a user of the classification of the software.	Ex. 1005 at 17:16-25 (emphasis added): Referring back to step 810, if the treated macro which corresponds to the targeted file is flagged as invalid, the treated macro is preferably not used to replace the infected macro. Thus, in step 835, alternative corrective action is taken by the file correcting module dependent upon how it is configured by the user. ***Various alternative corrective steps will be recognized such as notifying the user that the targeted file***

	includes a virus, removing the infected macro from the targeted file without replacement, or deleting the targeted file. *See also* Ex. 1007 at ¶ 103.

Claim 2	**Chen**
The method of claim 1, wherein the classification for the software includes malware.	Ex. 1005 at 15:43-54 (emphasis added): If it is determined in step 625 that the first instruction identifier is present, then the decoded macro is scanned in step 630 to determine whether the second instruction identifier is present. ***If in step 635 it is determined that the macro includes the second suspect instruction identifier, then in step 640 the decoded macro is flagged as infected by an unknown virus*** corresponding to the set of instruction identifiers. Information associating the decoded macro to the set of instruction identifiers that resulted in a positive unknown virus detection is stored in the data buffer 312 so that other modules such as the macro treating module 306 can treat the infected macro accordingly. *See also* Ex. 1007 at ¶ 102.

Claim 10	**Chen**
The method of claim 1, further comprising removing the software when the software is classified as malware or unwanted software.	Ex. 1005 at 16:24-47 (emphasis added): Referring back to step 710, if it is determined by the macro treating module 306 that a known virus is not present in the decoded macro, ***then the macro is treated to selectively remove an unknown virus***. The set of instruction identifiers that was used to detect an unknown virus in the decoded macro is available to the macro treating module 306 in the data buffer 312. An exemplary set includes first and second suspect instruction identifiers. In step 720, the first suspect instruction identifier is used to locate each suspect instruction which corresponds to the identifier. The decoded macros can be scanned to locate such instructions using the techniques described in connection with detecting the instructions used by the macro

	virus scanning module 304. If the instruction identifiers correspond to fragments of instructions rather than whole instructions, then the macro treating module 306 correlates each detected fragment to a whole instruction. The correlation will depend upon the programming language used by the macro. Conventional techniques can be used for the correlation. In step 725, the additional suspect instruction identifiers are used to locate corresponding additional suspect instructions. The located suspect instructions are then replaced in step 730. Similar to the replacement of known virus strings, preferably the suspect instructions are replaced with benign instructions. *See also* Ex. 1007 at ¶ 103.

Claim 13	**Chen**
The method of claims 1, further comprising blocking access to the software when the software is classified as malware or unwanted software.	Ex. 1005 at 16:24-47 (emphasis added): Referring back to step 710, if it is determined by the macro treating module 306 that a known virus is not present in the decoded macro, ***then the macro is treated to selectively remove an unknown virus***. The set of instruction identifiers that was used to detect an unknown virus in the decoded macro is available to the macro treating module 306 in the data buffer 312. An exemplary set includes first and second suspect instruction identifiers. In step 720, the first suspect instruction identifier is used to locate each suspect instruction which corresponds to the identifier. The decoded macros can be scanned to locate such instructions using the techniques described in connection with detecting the instructions used by the macro virus scanning module 304. If the instruction identifiers correspond to fragments of instructions rather than whole instructions, then the macro treating module 306 correlates each detected fragment to a whole instruction. The correlation will depend upon the programming language used by the macro. Conventional techniques can be used for the correlation. In step 725, the additional suspect instruction identifiers are used to locate corresponding additional suspect instructions. The located suspect instructions are then

	replaced in step 730. Similar to the replacement of known virus strings, preferably the suspect instructions are replaced with benign instructions. *See also* Ex. 1007 at ¶ 103.

VIII. CONCLUSION

With this Petition, Petitioner has established a reasonable likelihood that the challenged claims of the '344 Patent will be found unpatentable. Petitioner therefore respectfully requests that *inter partes* review of the '344 Patent be granted, and that claims 1, 2, 10, 13, 16 and 17 be held unpatentable.

Respectfully submitted,

Date: January 23, 2015

/ Jason Liu /

Jason Liu
Reg. No. 51,955
jasonliu@quinnemanuel.com
Attorney for Petitioner Fortinet Inc.

CERTIFICATE OF SERVICE
(37 C.F.R. §§ 42.6(e) and 42.105(a))

The undersigned hereby certifies that the above-captioned "PETITION FOR INTER PARTES REVIEW OF U.S. PATENT NO. 8,261,344," Power of Attorney and true copies of Exhibit Nos. 1001 to 1022 were served on January 23, 2015, at the official correspondence address for the attorney of record for the '344 Patent as shown in USPTO PAIR via EXPRESS MAIL:

STRATEGIC PATENTS P.C.
P.O. BOX 920629
NEEDHAM MA 02492

and

SEAN C. CUNNINGHAM
DLA PIPER LLP, US
401 B ST. STE. 1700
SAN DIEGO, CA 92101-4297

Date: January 23, 2015

/Jason Liu/
Jason Liu
Reg. No. 51,955
Quinn Emanuel Urquhart & Sullivan
50 California St, 22nd Floor
San Francisco, CA 94111
Telephone: (415) 875-6434
Facsimile: (415) 875-6000
Attorney for Petitioner Fortinet Inc.