UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

FORTINET, INC.,
Petitioner

v.

SOPHOS LIMITED AND SOPHOS INC.,
Patent Owner

Inter Partes Review No.: IPR2015-00618
U.S. Patent No.: 8,261,344 B2

PATENT OWNER'S RESPONSE
INTER PARTES REVIEW OF U.S. PATENT NO. 8,261,344

TABLE OF CONTENTS

TABLE OF EXHIBITS

EXHIBIT	DESCRIPTION
2001	Dr. Seth Nielson Deposition Exhibit: Copy of IPR2015-00618 Petition *(copy not provided herewith to avoid duplication)*
2002	Expert Declaration of Dr. Frederick B. Cohen
2003	Transcript of Dr. Seth Nielson Deposition (October 21, 2015)
2004	Brigham Young University 2005-2006 BS in Electrical Engineering MAP Sheet (https://registrar.byu.edu/advisement/pdf/05/393550.pdf)
2005	Brigham Young University Electrical and Computer Engineering Course Offerings 2002-2014 (https://www.ee.byu.edu/grad/courses)
2006	Brigham Young University Computer Science Graduate Policy Handbook Appendix (https://cs.byu.edu/graduate-policy-handbook-appendix)
2007	Excerpt of Brigham Young University Computer Science Courses (https://cs.byu.edu/courses)
2008	Brigham Young University CS 465 Course Description (https://wiki.cs.byu.edu/cs-465/course-information)
2009	Brigham Young University CS 665 Course Description (https://wiki.cs.byu.edu/cs-665/course-information)

Pursuant to 37 C.F.R. § 120, Patent Owner hereby submits this Response to the Petition seeking *inter partes* review of claims 1, 2, 10, 13, 16, and 17 (the "challenged claims") of U.S. Patent No. 8,261,344 (the "'344 patent"). This Response is timely filed in accordance with the Scheduling Order as modified by the Joint Stipulation dated October 8, 2015.

I. INTRODUCTION

The Board granted *inter partes* review of the '344 patent because it found the Petitioner established a reasonable likelihood of success in establishing that U.S. Patent No. 7,913,305 ("Bodorin") anticipates or makes obvious each of the challenged claims, either alone or in combination with U.S. Patent No. 7,373,664 ("Kissel") and U.S. Patent No. 7,448,084 ("Apap"). However, Bodorin falls far short of teaching the challenged claim elements. For example, Bodorin does not teach a library of gene information including a number of classifications based on groupings of genes. The Petitioner's argument does not even address this feature, which is unsurprising as it is entirely missing from Bodorin. Bodorin also fails to teach identifying functional blocks of software, as Bodorin examines computer behavior instead of blocks of code. Additionally, the combination of Bodorin, Kissel, and Apap is not one that would have been made by a person of ordinary skill in the art, and even if it were, Kissel and Apap do not compensate for the

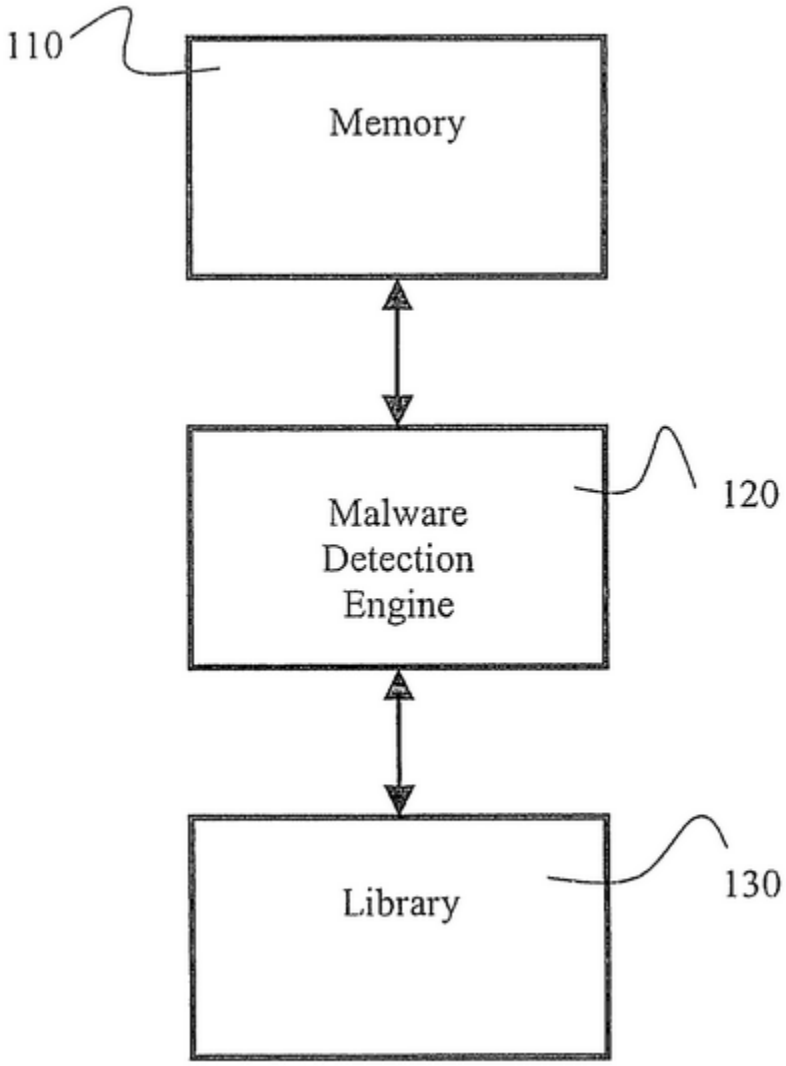
aforementioned deficiencies of Bodorin and do not teach or suggest testing a set of genes for false-positives against one or more reference files using a processor.

Accordingly, and for at least the reasons set forth below and explained in the Declaration of Dr. Frederick Cohen (Ex. 2002), the challenged claims of the '344 patent are neither anticipated by Bodorin nor obvious over Bodorin, either alone or in combination with Kissel and Apap. The validity of the challenged claims should be confirmed, and the *inter partes* review terminated.

II. OVERVIEW OF THE '344 PATENT

The '344 patent relates generally to identifying software by identifying software characteristics, or "genes," and matching the genes against classifications provided by groupings of genes. Ex. 1001 at 1:8-14. The '344 patent discloses a system and method for classifying software including identifying a functional block and a property of the software, identifying genes in the functional block and property, matching the genes against gene classifications, and classifying the software based on the classifications. *Id*. at 2:31-41.

In one example embodiment, illustrated below, a malware detection system 100 includes a memory 110, malware detection engine 120, and library 130. The system 100 may also include a user interface that is not illustrated. *Id*. at 3:18-24.



A specific method 200 for software classification by the system 100 is illustrated in Figure 2:

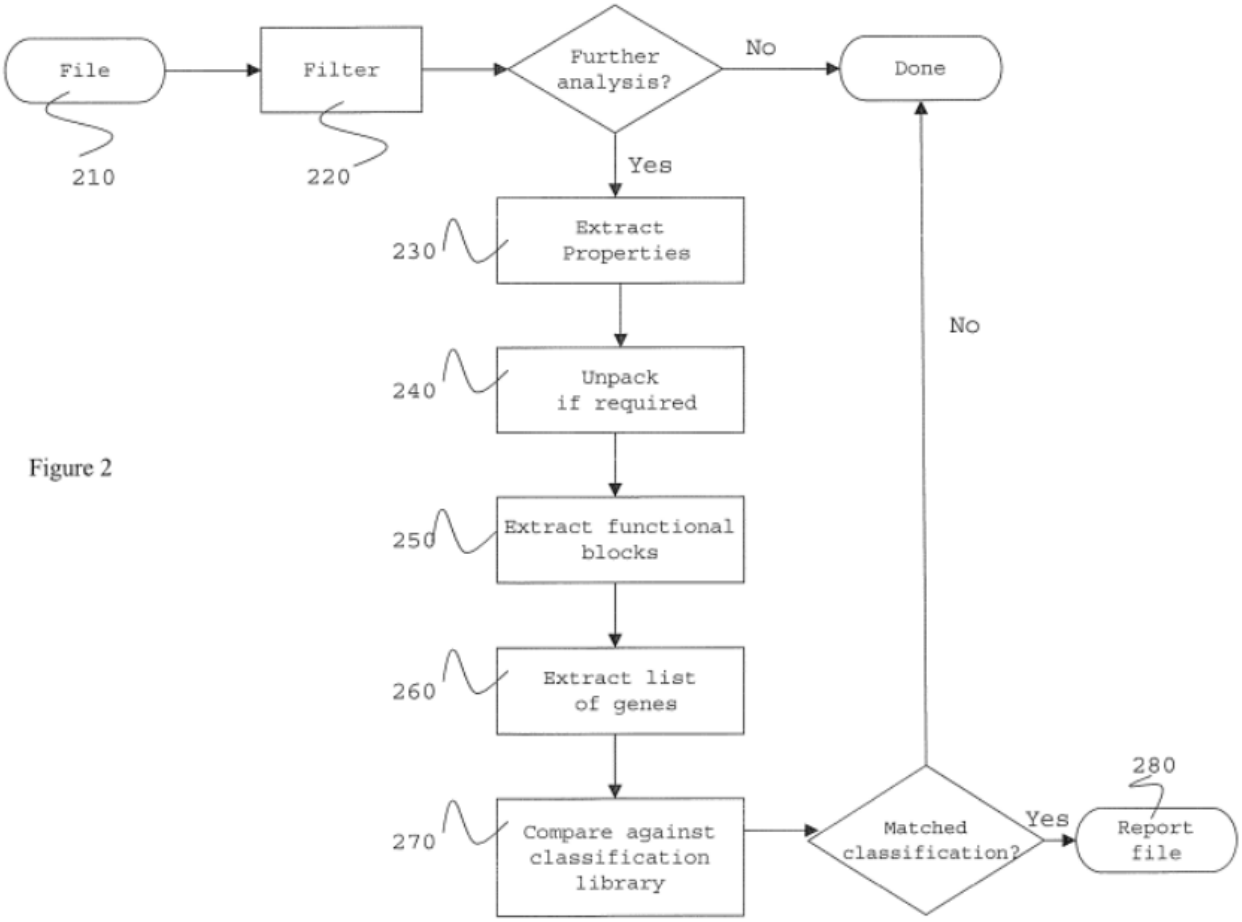


Figure 2

The analysis of a file begins at step 210. At step 220, a series of checks is performed to determine whether the file should be analyzed further (e.g., if the file is packed it may be more likely to contain malware and may be analyzed further, whereas if the file passes a checksum test it may not be checked). *Id*. at 4:60-5:3. If the file is to be analyzed further, properties are extracted from the file at step 230. Properties include file size or file name, for example. *Id*. at 5:4-6. After properties are extracted, the file is unpacked if necessary at step 240. *Id*. at 5:7-8. Next, at step 250, functional blocks are extracted from the file. Functional blocks include

sequences of application program interface (API) calls or string references, for example. The functional blocks are extracted from the file's binary code and illustrate the function and execution flow of the program. *Id*. at 5:17-22.

At step 260, the extracted functional blocks are searched for genes, and a list of genes present in the file is assembled. *Id*. at 5:29-30. A gene is described as "a piece of functionality or property of a program. *Id*. at 5:32-33. A piece of functionality is "described using sequences of APIs and strings, which can be matched against functional blocks." *Id*. at 5:33-35. The genes are compared against a library of classifications of genes at step 270. A classification of genes is detected for the file if all genes contained in the gene combination of the classification are found in the list of genes identified in the file. *Id*. at 5:46-50. For example, a functional block is shown in Ex. 1001 at 6:14-27. This functional block matches three genes, each of which has its own piece of functionality described by its own sequence of APIs and strings. *Id*. at 6:11-13, 6:28-42. The three genes together form a classification. *Id*. at 6:43-45. For example, gene "Runkey" has as a piece of functionality "set a run key" and includes the following sequence of APIs and strings:

- "software\microsoft\windows\currentversion\run"
- RegCreateKeyExA
- RegSetValueExA

Id. at 6:35-39.

Another example of a classification is as follows:

SockSend	Socket based activity
RunKey	Sets a run key
Exec	Executes other programs
CopySys	Copies itself to the system directory
AVList	Contains a list of AV (anti-virus) products
IRC	IRC references
Host	References the hosts file

Id. at 5:60-66. This classification includes seven genes and matches an IRC Bot malware file. *Id*. at 5:53-57.

An output, such as a report of significant identified classifications, is generated at step 280. *Id*. at 6:1-10.

III. CLAIM CONSTRUCTION

During an *inter partes* review, a claim should be given its broadest reasonable interpretation (BRI) in light of the specification. *See* 37 C.F.R. §42.100(b). Under the BRI, words of the claim must be given their plain meaning, unless such meaning is inconsistent with the specification. *In re Zletz*, 893 F.2d

319, 321, 13 USPQ.2d 1320, 1322 (Fed. Cir. 1989). The plain meaning of a term means the ordinary and customary meaning given to the term by those of ordinary skill in the art at the time of the invention. MPEP § 2111.01 ¶ III. *Phillips v. AWH Corp*., 415 F.3d 1303, 1312-13 (Fed. Cir. 2005) (*en banc*).

The Board construes "gene" as "a piece of functionality or property of a program," noting that "'APIs and strings' is still a limitation of the claims despite being omitted from the definition of 'gene.'" Decision at 6-7. The Decision interprets the limitation "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 and the at least one property of the software as a sequence of APIs and strings" to require a gene to describe either a functional block or a property, but not necessarily both. *Id*. at 8. Patent Owner respectfully disagrees with the Board that the '344 patent "assigns no special meaning or different meaning to property as opposed to functional block." *Id*. at 7. "Property" may not be explicitly defined, but examples of properties are given (e.g., file size or name). Ex. 1001 at 5:4-6. Indeed, Petitioner's expert witness, Dr. Seth Nielson, has explained how one of ordinary skill would understand "property" to mean something different from "piece of functionality" (i.e., a property "defines something about (the) program"). Ex. 2003 at 30:18-25. Accordingly, Patent Owner does not concede that the Board's construction of "gene" is proper under the BRI.

No other term was specifically construed by the Board. Other terms were instead "given their ordinary and customary meaning, as would be understood by one of ordinary skill in the art in the context of the entire disclosure." Decision at 6. Patent Owner and its expert have used the Board's construction of "gene" and the "ordinary and customary meaning" of the other claim terms in this Response.

IV. THE CHALLENGED CLAIMS ARE NOT ANTICIPATED OR OBVIOUS

A. Bodorin Does Not Anticipate Claims 1 and 2

Bodorin fails to disclose several elements of claim 1, as discussed below. It is well-established that a "claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros. v. Union Oil Co. of California,* 814 F.2d 628, 631 (Fed. Cir. 1987). Therefore, Ground 1 should be rejected and claim 1 determined to be patentable over Bodorin. Claim 2 depends from claim 1 and is valid for at least the reasons discussed below with respect to claim 1.

Bodorin discloses a "malware detection system that determines whether an executable code module is malware according to behaviors exhibited while [the code module is] executing." Ex. 1003 at Abstract. A block diagram of the malware detection system 200 is shown in Fig. 2.

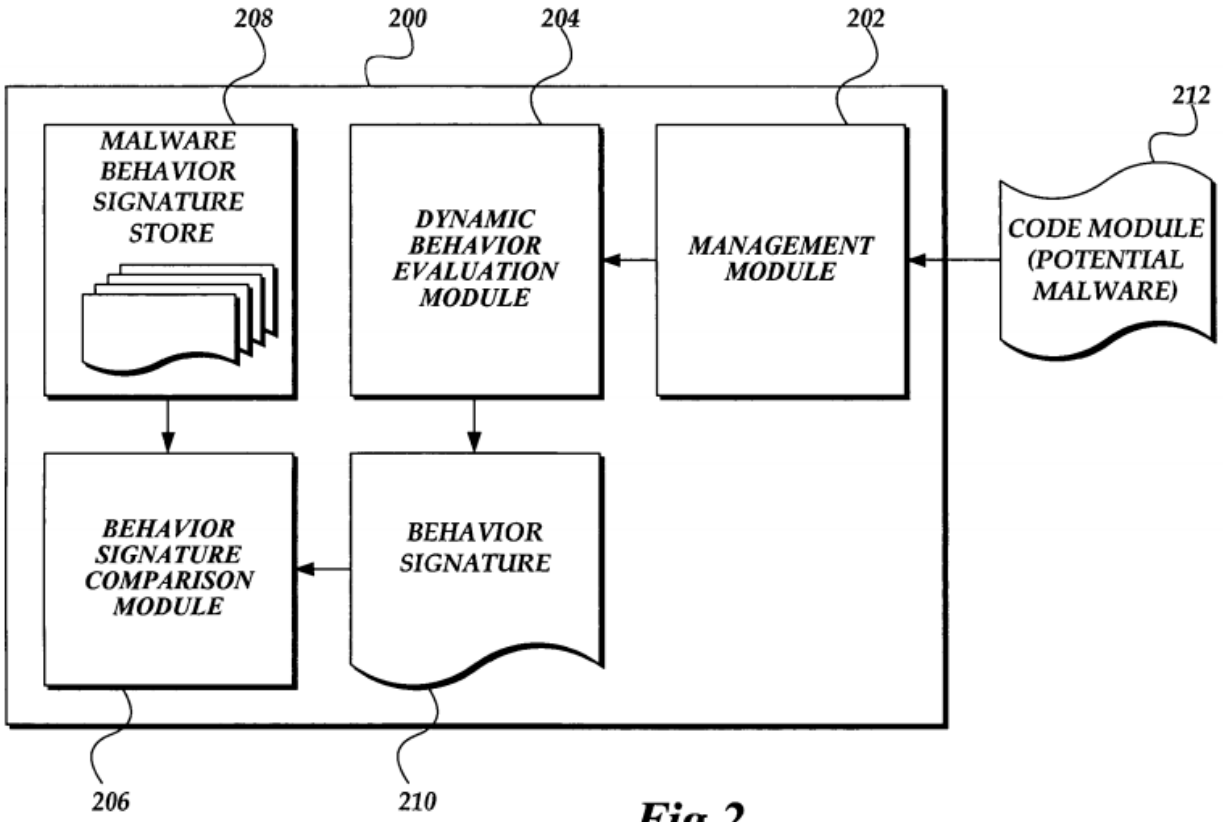


Fig.2.

The malware detection system 200 analyzes the code module 212 to identify malware within the module 212. The management module 202 first evaluates the code module 212 to determine the appropriate type of dynamic behavior evaluation module 204 needed to evaluate the behavior of the code module 212. *Id*. at 4:21-29. Once the appropriate dynamic behavior evaluation module 204 is determined, the code module 212 is handed over to the dynamic behavior evaluation module 204. The dynamic behavior evaluation module 204 executes the code module 212 and evaluates and records the code module's 212 dynamic behaviors. *Id*. at 4:33-50. Specifically, the dynamic behavior evaluation module 204 records "interesting"

behaviors exhibited by the code module 212 as the code module 212 executes. Interesting behaviors are behaviors potentially associated with malware. These interesting behaviors are compared against known malware behaviors. *Id*. at 4:51-58. Example interesting behaviors are shown in Table A of Bodorin (*Id*. at 5:5-20).

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)	

To make the comparison, the code module's 212 interesting behaviors are recorded in a file, which is called the behavior signature 210. The behavior signature comparison module 206 compares the behavior signature 210 against behavior signatures of known malware. The known malware behavior signatures are based on exhibited behaviors of malware. *Id*. at 5:21-43. If the behavior

signature comparison module 206 completely matches the behavior signature 210 against a known malware behavior signature, the malware detection system 200 reports the code module 212 as malware. *Id*. at 5:44-49.

1. Bodorin does not disclose "providing a library of gene information including a number of classifications based on groupings of genes"

Claim limitation 1[a] recites "providing a library of gene information including a number of classifications based on groupings of genes." Patent Owner respectfully submits that Bodorin fails to anticipate claim 1 at least because Bodorin does not disclose providing a library of gene information including a number of classifications based on groupings of genes. Ex. 2002 at 16-18.

As noted above, "gene" is being construed as "a piece of functionality or property of a program" in this *inter partes* review. Thus, the claimed library of gene information must include a number of classifications based on groupings of pieces of functionality or properties of a program. A "piece of functionality is described using sequences of APIs and strings." Ex. 1001 at 5:33-34. As Patent Owner's expert notes, a single API and an associated string or strings is not a sequence of APIs and strings, because a sequence is a plurality of APIs and strings. Ex. 2002 at 24. The '344 patent presents examples of "functionality" as sequences of APIs and strings that define a software characteristic. *Id*. at 6:50-55. For example, "CopySys" is a sequence of five APIs and strings defining a copy

functionality. *Id.* at 6:28-34, 6:52-55. Each of the lines in the CopySys gene contributes to a description of the overall functionality of the gene. This is evident from a reading of Ex. 1001 at 6:30-34 and is what one of ordinary skill in the art would understand according to Petitioner's own expert, Dr. Nielson: "[CopySys is] identifying multiple steps that would be involved in a copy to the system folder." Ex. 2003 at 33:7-13. A classification, then, is a grouping of genes including such pieces of functionality (i.e., sequences of APIs and strings). For example, genes "'SockSend, RunKey, Exec, CopySys, AVList, IRC, and Host' are combined to form a classification for an 'IRC Bot' program." Ex. 1001 at 6:57-60. The classification is a combination of genes, and each gene in turn is a sequence of APIs and strings that constitute a piece of functionality of a program.

In contrast to the '344 patent, Bodorin teaches classifications based on groupings of APIs and strings (i.e., classifications based on functionalities, rather than on groupings of functionalities, at best). Bodorin teaches identifying interesting behaviors exhibited by an executing code module 212 and storing the identified interesting behaviors in a behavior signature 210. Ex. 1003 at 21-23. FIGS. 5A and 5B show example behavior signatures 500 and 512. Each interesting behavior is represented as a behavior token 502, and each behavior (interesting or otherwise) may be accompanied by one or more parameter values 504, 506, 508, which may include strings as in 508. *Id*. at 7:16-46. Indeed, Petitioner notes that

the "behaviors monitored in Bodorin include some of the very same API calls that are the bases for 'genes' in the '344 patent." Petition at 20 (emphasis added). These individual API calls are combined into behavior signatures and compared with stored behavior signatures. Ex. 1003 at 5:27-43. The Petition does not show that Bodorin discloses, teaches, or suggests using sequences of API calls and strings to denote interesting behaviors. Thus, claim 1 requires three distinct categories of information. Ex. 1001 at 6:11-42, 7:60-67. See also Ex. 2002 at 16-18 and Ex. 2003 at 31:2-33:13. Bodorin, by contrast, only discloses two (interesting behavior, behavior signature), and does not disclose (or even suggest) anything equivalent to the third category, the claimed "classifications." Ex. 1003 at 5:27-43.

The '344 patent does provide an example of at least one gene that includes a single API: "Exec," which includes the API "CreateProcessA." Ex. 1001 at 6:40-42. This does not contradict the arguments presented above, as Applicant explicitly disclaimed such single-API genes in the amendment filed April 3, 2012 by specifically defining a gene as "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 55 (emphasis added). Applicant explained that using sequences of APIs and strings to describe software was an inventive concept. *Id*. at 59. The Examiner specifically noted the feature "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" as being instrumental in overcoming the prior art. *Id*. at 15. Thus, the distinctions between Bodorin and the '344 patent are reinforced by the prosecution history of the '344 patent.

Additionally, despite the conclusory assertions of Petitioner (Petition at 4), there is no disclosure, teaching, or suggestion anywhere in Bodorin that the "behaviors" recorded by the dynamic behavior execution module 204 are equivalent to "pieces of functionality" of a program. The behaviors are exhibited by a code module 212 as it executes within a dynamic behavior evaluation module 204. Ex. 1003 at 4:51-54. As those of ordinary skill in the art will appreciate, "behavior" and "functionality" are not equivalent terms. Ex. 2002 at 16-17. "The same program may demonstrate different behaviors in different environments," so the same program, with the same underlying functionality, may exhibit different behaviors in different environments. *Id*. Accordingly, the APIs and strings forming "interesting behaviors" in Bodorin do not describe behaviors of the code module 212 alone, but instead describe behaviors of the code module 212 executing in the dynamic behavior evaluation module 204. This behavior is not inherent to the code module 212 as it requires the dynamic behavior evaluation module 204 and is affected by the specific environment provided by the dynamic behavior evaluation module 204. *Id*.

Claim limitation 1[a][1] requires a number of classifications based on groupings of genes. Genes are pieces of functionality or properties of programs. Pieces of functionality are sequences of APIs and strings. Thus, claim limitation 1[a] requires three layers of organization – APIs/strings, genes, and classifications. Bodorin teaches, at most, two. Accordingly, Bodorin does not anticipate claim limitation 1[a]. Because claim limitations 1[d], 1[e], and 1[f] also require classifications, Bodorin does not anticipate elements 1[d], 1[e], and 1[f], as well. Ex. 2002 at 24-26.

2. Bodorin does not disclose "identifying at least one functional block and at least one property of the software"

Claim limitation 1[b] recites "identifying at least one functional block and at least one property of the software". Patent Owner respectfully submits that Bodorin fails to anticipate claim 1 at least because Bodorin does not disclose identifying at least one functional block of the software. Ex. 2002 at 18-24.

The '344 patent teaches that functional blocks include sequences of API calls and strings that illustrate the function and execution flow of a program. Ex. 1001 at 5:17-20. The functional blocks are extracted from a program file's binary code. *Id*. at 5:21-22. The experts for the Petitioner and Patent Owner agree that a functional block is a segment of a program or "chunk" of software with

[1] The labeling of the claims as set out in the Petition is used herein for consistency.

recognizable boundaries. Ex 1007 at ¶62, Ex. 2002 at 18-19, Ex. 2003 at 35:4-36:11. This is consistent with the '344 patent's teaching that functional blocks are extracted from a program file's binary code.

The Petition does not identify any disclosure in Bodorin that identifies segments or chunks of a program. Bodorin watches code as it executes and identifies interesting behaviors. Ex. 1003 at 4:51-54. The Petition does not identify what specifically constitutes a "functional block" in Bodorin, instead merely block quoting Bodorin at 4:51-5:20 for the entire clause 1[b]. Petition at 23. Ex. 1003 at 4:51-5:20 is silent on identifying a segment or chunk of a program, let alone one that illustrates the function and execution flow of the program. Petitioner's expert does state that the identified behaviors include API calls that illustrate the function and execution flow of the program (Ex. 1007 at ¶62), but behaviors that illustrate function and execution flow are not program segments that illustrate function and execution flow. Likewise, identifying behaviors is not identifying functional blocks. Ex. 2002 at 19-20 and 23-24. Identifying the functional block is identifying a program segment that can illustrate the function and execution flow of the program. Identifying the behavior is, in fact, identifying the function of the program and using that information to make an observation about the program (i.e., identify the interesting behaviors). *Id*. Indeed, Petitioner's expert explains that programs written in different languages may produce the same interesting

behaviors despite having completely different code (and therefore different functional blocks). Ex. 2003 at 42:14-48:9.

Furthermore, experts for the Petitioner and Patent Owner agree that for "execution flow" to make sense, more than one instruction must be involved (e.g., a series of instructions, a loop that repeats one or more instructions (thus being essentially those repeated instructions in sequence), or a branch instruction specifying where the program will go to find a next instruction). Ex. 2002 at 19-20, Ex. 2003 at 35:20-36:11. Even if the "interesting behaviors" of Bodorin were identified functional blocks of code, they would not illustrate the execution flow. Bodorin teaches that each interesting behavior is represented as a single behavior token and, optionally, one or more parameter values. Ex. 1003 at 7:23-27. Only interesting behaviors are recorded, and other behaviors are not recorded. *Id*. at 8:12-16. In the '344 patent, a functional block includes multiple instructions in context, illustrating an execution flow. Ex. 1001 at 6:11-42. In Bodorin, individual behaviors are isolated out of context and therefore cannot illustrate an execution flow.

Claim limitation 1[b] requires identification of a functional block, which is a sequence of API calls and strings extracted from a program's code that illustrates the function and execution flow of the program. Bodorin's interesting behaviors are not functional blocks and they do not illustrate the execution flow of a

program. Ex. 2002 at 19-20. Accordingly, Bodorin does not anticipate claim limitation 1[b].

3. Bodorin does not disclose "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"

As discussed above, Bodorin does not disclose identifying at least one functional block. Thus, even if a single API with optional string(s) used to identify interesting behaviors is regarded as a gene (which Patent Owner does not concede), it does not describe an identified functional block. Furthermore, a single API with a string (or even multiple strings) does not describe a program flow as discussed above. Finally, a single API and an associated string or strings is not a sequence of APIs and strings, because a sequence is a plurality of APIs and strings. Ex. 2002 at 24. Accordingly, Bodorin does not anticipate claim limitation 1[c].

B. Bodorin Does Not Render Claims 10 and 13 Obvious

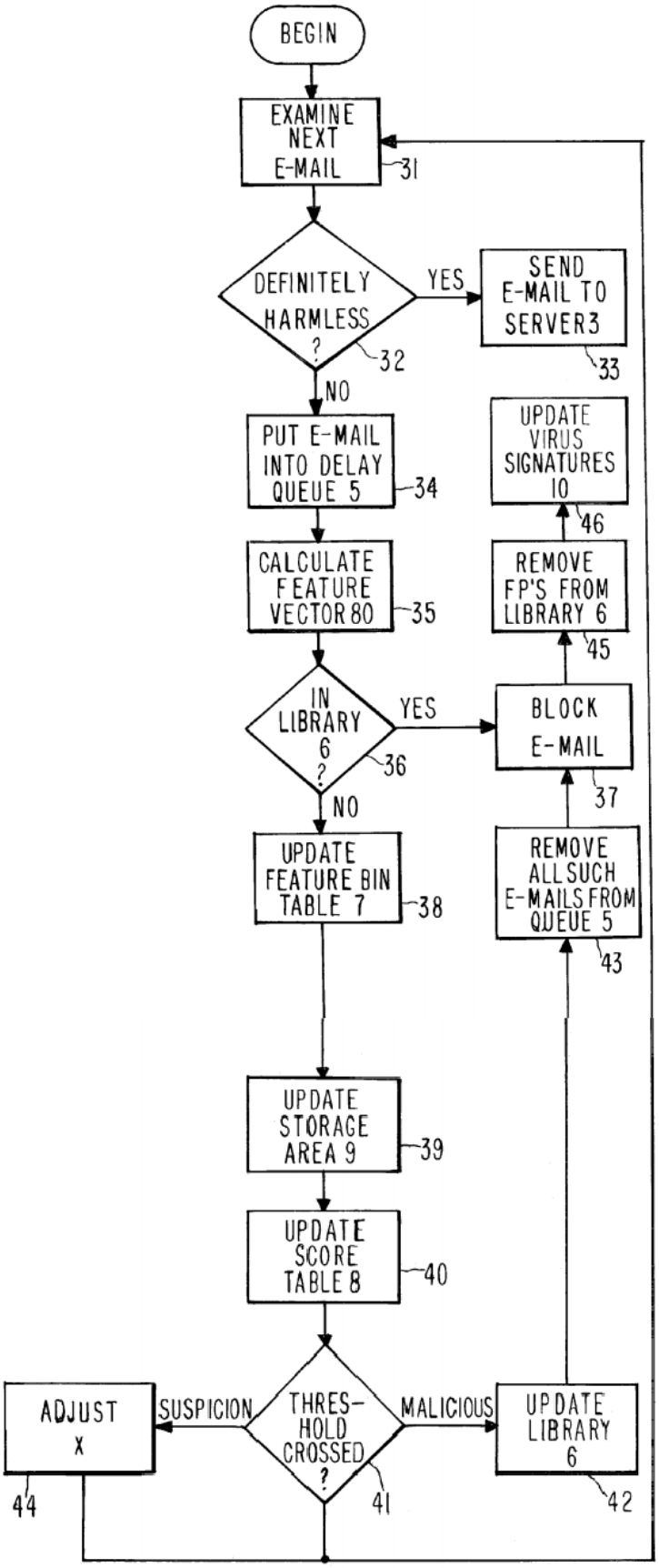
Claim 1 and its dependent claims are not anticipated by Bodorin for at least the reasons presented in section IV(A), and claim 1 is therefore valid. As claims 10 and 13 depend on valid claim 1, they are also valid. Ground 2 should be rejected and claims 10 and 13 confirmed for at least this reason.

Furthermore, Bodorin does not explicitly teach "removing the software when the software is classified as malware or unwanted software" (claim 10) or "blocking access to the software when the software is classified as malware or unwanted software," (claim 13) as acknowledged by Petitioner. Petition at 27. Petitioner cites to Bodorin's description of prior art anti-virus software including these features, noting that Bodorin explains that the malware detection system described therein may be used in conjunction with current anti-virus software. Petition at 27-29. Bodorin teaches that in such cases, the anti-virus software's signature scanning techniques should be used as a first step before turning to the described malware detection system. Ex. 1003 at 4:8-12. This argument may explain a motivation to combine anti-virus scanning (which includes virus removal and blocking) and the system of Bodorin in the described order, but the argument does not establish a motivation to re-run the virus scanner for removal and blocking after using the system of Bodorin. See also Ex. 2002 at 27-31. Indeed, given that a person of ordinary skill in the art as defined by the Petitioner's expert is a person of relatively low skill (e.g., having a Master's degree in electrical engineering and potentially no relevant background or training in computer security or malware detection), it is doubtful that such a motivation can be found in Bodorin. Ex. 2002 at 31 and 11-12, Ex. 2004, Ex. 2005, Ex. 2006, Ex. 2007, Ex. 2008, Ex. 2009, Ex. 2003 at 18:23-25, Section IV(C)(1) *infra*.

C. Bodorin In View Of Kissel And Apap Does Not Render Claims 16 and 17 Obvious

Bodorin in view of Kissel And Apap fails to teach several elements of claim 16, as explained below. Additionally, Petitioner's proposed combination of Bodorin, Kissel, and Apap is not obvious to one of ordinary skill in the art. Thus, claim 16 is valid. As claim 17 depends from claim 16, claim 17 is valid for at least the reasons discussed below with respect to claim 16. Ground 3 should be rejected and claims 16 and 17 confirmed.

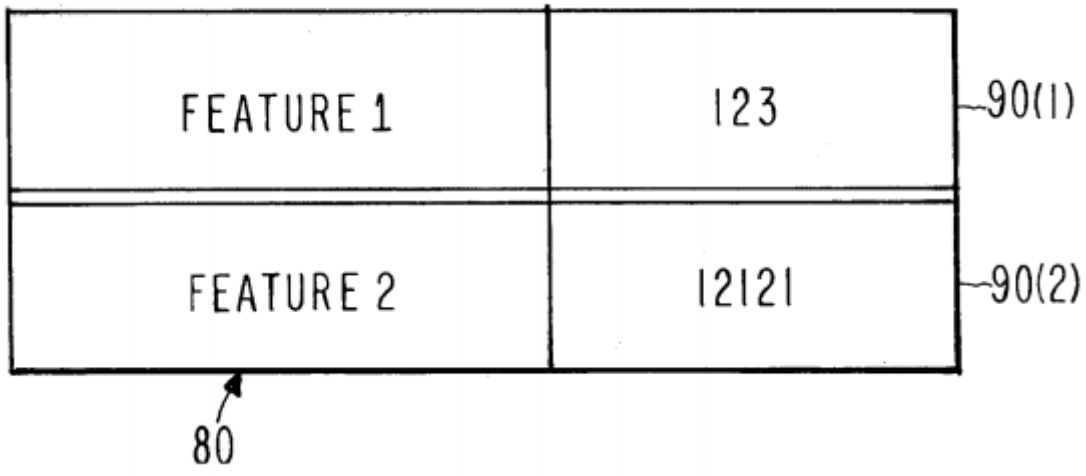
Kissel discloses methods and systems "for detecting the presence of malicious computer code in a plurality of e-mails." Ex. 1004 at Abstract. Kissel's system operation is shown in Figs. 3A-3B.


BEGIN
EXAMINE NEXT E-MAIL
31
DEFINITELY HARMLESS ?
32
YES
SEND E-MAIL TO SERVER3
33
NO
PUT E-MAIL INTO DELAY QUEUE 5
34
CALCULATE FEATURE VECTOR 80
35
IN LIBRARY 6 ?
36
YES
BLOCK E-MAIL
37
NO
UPDATE FEATURE BIN TABLE 7
38
UPDATE STORAGE AREA 9
39
UPDATE SCORE TABLE 8
40
THRES-HOLD CROSSED ?
41
SUSPICION
ADJUST X
44
MALICIOUS
UPDATE LIBRARY 6
42
REMOVE ALL SUCH E-MAILS FROM QUEUE 5
43
REMOVE FP'S FROM LIBRARY 6
45
UPDATE VIRUS SIGNATURES 10
46

Emails are examined at step 31 as they arrive at a gateway 2. If an email is judged to be definitely harmless at step 32, it is sent to the server at step 33. If not, it is put into a delay queue 5 for a prescribed amount of time at step 34. Then, a feature vector 80 is calculated at step 35. The feature vector 80 contains feature instance entries 90. If a feature instance 90 corresponds to a blocked feature instance deemed to be indicative of malicious code in library 6 at step 36, the email is blocked at step 37. Ex. 1004 at 5:1-60. If the feature instance 90 does not correspond to a known blocked feature instance, additional processing is performed to determine if the feature entry 90 crosses a suspicious or malicious threshold at steps 38-41. If the malicious threshold is crossed, the library 6 is updated at step 42 with the feature instance 90. Ex. 1004 at 6:7-54.

Example feature instances 90(1) and 90(2) within a feature vector 80 are shown in Fig. 4.

FIG. 4



The feature instances 90 are hashes calculated from features of an email. Ex. 1004 at 4:49-62 and 5:25-30. Email features that can be tracked by Kissel include the following:

> 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.
> Ex. 1004 at 3:34-46

In the case of parsed script (and a hash derived therefrom), code between <SCRIPT> tags is examined and parsed to generate a tokenized representation of the script. The tokenized representation includes keywords (e.g., var* and if*), but removes the bulk of the code. Ex. 1004 at 3:47-4:3.

Apap discloses "a method for detecting intrusions in the operation of a computer system". Ex. 1006 at Abstract. Petitioner cites Apap for teaching generating a model for normal behavior, employing an algorithm for detecting

anomalous behavior, and training and testing the algorithm by comparing detected anomalous behavior against normal behavior to generate detection and false positive statistics. See page 37 of the Petition and Ex. 1006 at 5:10-20 and 15:44-16:20.

1. A person of ordinary skill in the art would not combine Bodorin, Kissel, and Apap

The parties agree that a person of ordinary skill is fairly unsophisticated, being a person having "a Master's degree in electrical engineering, computer engineering or computer science; or a Bachelor's degree in electrical engineering, computer engineering or computer science with at least two years experience working with hardware and software for computer and network security." Ex. 1007 at ¶21, Ex. 2002 at 10. Such a person (e.g., a person having a Master's degree in electrical engineering and no relevant experience) may be almost completely unfamiliar with computer security in some cases. For example, Dr. Cohen's discussion of the requirements for an example Master's degree in electrical engineering illustrate that it would have been possible for such a person to have had no formal training in computer security whatsoever at the time the '344 patent was filed. Ex. 2002 at 11-12, Ex. 2004, Ex. 2005, Ex. 2006, Ex. 2007, Ex. 2008, Ex. 2009. That is, one could have graduated with a Master's degree in Electrical Engineering having taken no computer security courses. Ex. 2002 at 12. Dr.

Nielson agrees that "there would be some person with a master's degree in electrical engineering out there somewhere who could not do this." Ex. 2003 at 18:23-25.

The combination of Bodorin with Kissel and Apap must be evaluated with this person of ordinary skill in mind. As both experts note, "Bodorin does not disclose updating the malware behavior signature store by testing the generated behavior signature for false positives and storing the tested generated behavior signature," which is claimed in claim 16. Ex 1007 at 91, Ex. 2002 at 32. The Petition asserts that a person of ordinary skill would find it obvious to look to Kissel and Apap for a way to update the behavior signature store 208 of Bodorin. Petition at 44. However, a person of ordinary skill in the art as set forth in the Petition "need have no background in computer security or malware detection, and would have to somehow relate monitoring Windows registry access… to worm and spam detection, which are largely about network-based events rather than within-processor events… and relate this to dynamic code analysis, which has nothing obvious to do with Windows registry access or detection of network-related spam and worm attacks… to even consider such a combination." Ex. 2002 at 33. This is true in spite of the fact that a person with a Master's degree in electrical engineering may have no knowledge of any of these topics, and indeed may be unaware of these topics altogether. *Id*.

Petitioner's asserted motivation to combine appears to be based not on the point of view of a person of ordinary skill in the art, but on a maker of anti-virus software. Petition at 44-45, citing Ex. 1007 at ¶¶91-94.

> 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. 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. (sic)
>
> Ex. 1007 at ¶93.

A maker of anti-virus software is not a person with a Master's degree in electrical engineering, but rather a person who is routinely engaged (and presumably particularly knowledgeable) in developing malware detection systems and methods. Such a person may make the combination of art suggested by Petitioner, but a person of ordinary skill "is not asserted to be or shown to be or have the skills or motivations of 'makers of anti-virus software.'" Ex. 2002 at 41.

When the knowledge of a person of ordinary skill in the art is considered, the combination of Bodorin, Kissel, and Apap offered by Petitioner is apparent in hindsight only. Ex. 2002 at 33. It is well established that improper hindsight cannot be the basis for an obviousness rejection. MPEP § 2145. The combination is not obvious, and the Grounds based on the combination should be rejected and claims 16 and 17 deemed patentable.

2. Bodorin in view of Kissel and Apap does not teach or suggest "providing a library of gene information including a number of classifications based on groupings of genes"

Even if the combination of Bodorin, Kissel, and Apap is within the reach of a person of ordinary skill (which Patent Owner does not concede), it does not teach or suggest every element of claim 16. As discussed above, Bodorin does not teach "providing a library of gene information including a number of classifications based on groupings of genes." Ex. 2002 at 42. Nothing in the Petition alleges that Kissel or Apap compensates for this deficiency, and the Petition does not cite to Kissel or Apap for this element. Petition at 45. Indeed, neither Kissel nor Apap teaches anything about providing a library of any kind, particularly one including a number of classifications based on groupings of genes. Accordingly, claim 16 is valid for at least the reason that claim limitation 16[a] is not obvious over Bodorin in view of Kissel and Apap. Ex. 2002 at 42.

3. Bodorin in view of Kissel and Apap does not teach or suggest "identifying one or more genes each describing a functionality or property of the software as a sequence of APIs and strings"

As discussed above, Bodorin does not teach "identifying at least one functional block" and "identifying one or more genes each describing one or more of the at least one functional block… as a sequence of APIs and strings." Ex. 2002 at 42. Furthermore, a functional block is a segment of a program (Ex 1007 at ¶62, Ex. 2002 at 18-19, Ex. 2003 at 35:4-36:11), and a functionality is extracted from a functional block (and is thus a subset thereof). Ex. 1001 at 5:29-32. "API function calls are defined functionality of the software." Ex. 2003 at 61:19-21. As Bodorin does not identify a functional block of the software, Bodorin does not identify a functionality that is a subset of a functional block. Additionally, a single API and an associated string or strings is not a sequence of APIs and strings, because a sequence is a plurality of APIs and strings. Ex. 2002 at 24. Ex. 2002 at 19-20 and 23-24.

The Petition does not allege that Kissel or Apap compensates for this deficiency, and does not cite to Kissel or Apap for this element. Petition at 46-47. Indeed, neither Kissel nor Apap teaches anything about identifying anything describing a functionality or property of software as a sequence of APIs and

strings. Accordingly, claim 16 is valid for at least the reason that claim limitation 16[b] is not obvious over Bodorin in view of Kissel and Apap.

4. Bodorin in view of Kissel and Apap does not teach or suggest "combining a plurality of genes that describe the software, thereby providing a set of genes"

As discussed above, Bodorin does not teach "identifying one or more genes each describing a functionality or property of the software as a sequence of APIs and strings" or "providing a library of gene information including a number of classifications based on groupings of genes." Ex. 2002 at 42. Instead, interesting behaviors are identified and combined into behavior signatures. The behavior signatures are not a grouping of sequences of APIs and strings. Ex. 1003 at 5:27-43, Ex. 2002 at 16-18 and 24.

The Petition does not allege that Kissel or Apap compensates for this deficiency, and does not cite to Kissel or Apap for this element. Petition at 47. Indeed, neither Kissel nor Apap teaches anything about grouping separate elements describing a functionality or property of software as a sequence of APIs and strings. Accordingly, claim 16 is valid for at least the reason that claim limitation 16[c] is not obvious over Bodorin in view of Kissel and Apap.

5. Bodorin in view of Kissel and Apap does not teach or suggest "testing the set of genes for false-positives against one or more reference files using a processor"

As Petitioner notes, Bodorin does not teach or suggest "testing the set of

genes for false-positives against one or more reference files using a processor." Petition at 43. Kissel and Apap are cited for allegedly teaching these features. However, they do not.

Kissel teaches a screening method that identifies email features and determines whether they match entries in a features library 6. Ex. 1004 at 5:20-35. If an email contains no features corresponding to entries in the library 6, the library is updated with entries for the new features identified in the email. *Id*. at 6:7-10. The entries generated by the method of Kissel include a message ID, a timestamp, and a score which may or may not be above a malicious threshold. Ex. 1004 at 4:58-65 and 6:20-23. In contrast, Bodorin teaches that the malware behavior signature store 208 is periodically updated with new behavior signatures. Ex. 1003 at 6:1-5. The behavior signatures in the store 208 represent behaviors exhibited by a code module and represented as a behavior token 502 and one or more parameter values 504 and 506, as shown in FIGS. 5A and 5B. Ex. 1003 at 7:16-27. The entries of Kissel bear no relation to the behavior signatures of Bodorin and are therefore unrelated to updating a library such as that disclosed in Bodorin (and, indeed, unrelated to the claimed library of gene information). Ex. 2002 at 43-44. Kissel, in essence, is cited for teaching updating something, but Kissel does not update the same kind of data that is stored in the behavior signature store 208 of Bodorin. Ex. 2002 at 43-47.

Apap does not compensate for these deficiencies. Apap teaches examining stored registry access data directly to train an algorithm that will subsequently be used to monitor registry access directly in real time. Ex. 1006 at 10:1-11. Apap is only cited for evaluating for false positives over "normal data," which Petitioner regards as a reference file. Ex. 1006 at 15:44-51, Ex. 1007 at ¶85, Ex. 2003 at 65:5-66:14. Apap does not teach or suggest that the stored registry access data is a gene or anything resembling a gene. Ex. 2002 at 44.

Furthermore, none of Bodorin, Kissel, or Apap disclose, teach, or suggest testing for false positives with a processor. The Petition does not cite to any teaching of testing for false positives with a processor anywhere in any reference. Petition at 47-49. Indeed, Kissel discloses "manual setting of the scoring parameters," directly refuting the use of a processor. Ex. 1004 at 8:56-62, Ex. 2002 at 44. It is possible to test for false positives without the use of a processor. Ex. 2003 at 68:17-24.

For at least the foregoing reasons, claim 16 is valid because claim limitation 16[d] is not obvious over Bodorin in view of Kissel and Apap.

V. CONCLUSION

For the foregoing reasons, Patent Owner respectfully submits that the Board should not find any claim of the '344 patent unpatentable based on any ground set forth in the Petition. The Petitioner has not met its burden to establish anticipation of any claim of the '344 patent. Moreover, the Petitioner has not met its burden to establish obviousness of any claim of the '344 patent.

Dated: October 28, 2015

Respectfully Submitted,

/ James M. Heintz /

James M. Heintz
Lead Counsel for Patent Owner
Reg. 41,828
SophosFortinetIPR@dlapiper.com
Phone: 703-773-4148
Fax: 703-773-5200

Gianni Minutoli
Reg. No. 41,198
gianni.minutoli@dlapiper.com
Phone: 703-773-4045
Fax: 703-773-5200

Nicholas Panno
Reg. No. 68,513
nicholas.panno@dlapiper.com
Phone: 703-773-4157
Fax: 703-773-5200

Postal and Hand Delivery Address:
DLA Piper LLP (US)
11911 Freedom Drive, Suite 300
Reston, VA 20190-5602

Attorneys for Patent Owner Sophos Ltd. and Sophos Inc.

CERTIFICATE OF SERVICE

The undersigned hereby certifies that a true copy of the foregoing instrument was served on Petitioner, Fortinet Inc., by emailing a copy to counsel at the email addresses listed below:

Jason Liu
jasonliu@quinnemanuel.com

Jordan R. Jaffe
jordanjaffe@quinnemanuel.com

IPR-FortinetSophos@quinnemanuel.com

Dated: October 28, 2015

/James M. Heintz/
James M. Heintz
Reg. 41,828
Jim.heintz@dlapiper.com
Counsel for Patent Owner