

Fortinet-1007

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
PATENT TRIAL & APPEAL BOARD**

In re Patent of: Ben Godwood et al.
U.S. Patent No.: 8,261,344
Issue Date: September 4, 2012
Appl. No.: 11/428,203
Filing Date: June 30, 2006
Title: Method and System for Classification of Software Using
Characteristics and Combinations of Such Characteristics

DECLARATION OF DR. SETH NIELSON, Ph.D.

(1.) I am Dr. Seth Nielson. I submit this report on behalf of Fortinet, Inc. in connection with its request for *inter partes* review of U.S. Patent No. 8,261,344 (“the ‘344 Patent”).

I. INTRODUCTION

A. Qualifications

(2.) I am a Principal at Harbor Labs in Baltimore, Maryland. I received a B.S. in Computer Science in 2000 and my M.S. in Computer Science in 2004, both from Brigham Young University in Provo, UT. I received my Ph.D. in Computer Science in 2009 from Rice University in Houston, TX. My doctoral dissertation concerned “Designing Incentives for Peer-to-Peer Systems.” I am the recipient of the Brown Fellowship and a Graduate Fellowship from the Rice University Computer Science Department. I was also a John and Eileen Tietze Fellow.

(3.) From 2001 through 2003, I worked as a software engineer at Metrowerks (formerly Lineo, Inc.). There I gained substantial experience in software architecture, computer networking, and technical project management. In particular, I developed and maintained the GUI for the Embedix SDK, ported the Linux GUI of the Embedix SDK to Windows, created an automated system to forward Linux python scripts to a Windows GUI, and developed a packaging and automated updating system for client software.

(4.) In 2005, I completed an internship at Google, where I designed and implemented a solution to privacy loss in Google Web Accelerator. Through this project, I worked extensively on web caching technologies and developed expertise in network communications and network security.

(5.) From 2005 through 2011, I served as a Security Analyst and later a Senior Security Analyst for Independent Security Evaluators. There, I developed a parallel-processing based security tool, developed a FIPS-certified encryption library, developed hardware-accelerated encryption algorithms, developed encrypted file-system prototypes, developed an encryption library for an ISE client, performed port-scanning analyses, evaluated security protocols using formal methods and hand analysis, and evaluated security failures.

(6.) In 2011, I began work as a Research Scientist at Harbor Labs. I am now a Principal, specializing in network security, network communications,

software architecture, and programming languages. I have analyzed an extensive collection of commercial software, including software related to secure email, cloud-based multimedia delivery, document signing, anti-virus and anti-intrusion, high-performance routing, networking protocol stacks in mobile devices, PBX telecommunications software, VoIP, and peer-to-peer communications. I have also analyzed security considerations for potential technology acquisitions, re-created heuristic signatures for 1995-era viruses, and re-created a 1995-era network for testing virus scanners of that time period in gateway virus scanning.

(7.) In 2014 I received an appointment as a Lecturer at Johns Hopkins University and taught a graduate level course in Network Security. I am now an Associate Assistant Research Scientist. My responsibilities include teaching courses and mentoring students. For the Network Security class that I teach, I created the entire curriculum from scratch and developed a novel laboratory framework for student experimentation. I covered topics ranging from cryptography and access controls to network architecture and user psychology.

(8.) In addition to my course instruction, I also mentor Masters students at Johns Hopkins in their capstone projects. These projects include networking security and privacy concerns across a wide range of technologies.

(9.) I make this declaration based on personal knowledge and I am competent to testify about the matters set forth herein. A copy of my latest *curriculum vitae* (CV) is attached to this declaration as Appendix A.

B. Basis of My Opinion and Materials Considered

(10.) I have reviewed the '344 Patent and its file history. I have reviewed the prior art and other documents and materials cited herein. My opinions are also based in part upon my education, training, research, knowledge, and experience. For ease of reference, the full list of information that I have considered is included in Appendix B.

C. Understanding of Legal Standards

1. Anticipation

(11.) I understand a patent claim is “anticipated” if each and every limitation of the claim is disclosed in a single prior art reference. Section 102 of the Patent Statute was amended on March 16, 2013. The earlier version of Section 102 applies to the patent at issue given its filing date.

(12.) Each element of a patent claim may be disclosed by a prior art reference either expressly or inherently. Further, my understanding is that even an “express” disclosure does not necessarily need to use the same words as the claim. An element of a patent claim is inherent in a prior art reference if the element must necessarily be present and such would be recognized by a person of ordinary skill

in the art. However, I understand that inherency cannot be established by mere probabilities or possibilities.

2. Obviousness

(13.) A patent claim is invalid if the differences between the patented subject matter and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person of ordinary skill in the art. I am informed that this standard is set forth in 35 U.S.C. § 103(a).

(14.) When considering the issues of obviousness, I am to do the following: (i) determine the scope and content of the prior art; (ii) ascertain the differences between the prior art and the claims at issue; (iii) resolve the level of ordinary skill in the pertinent art; and (iv) consider objective evidence of non-obviousness. I appreciate that secondary considerations must be assessed as part of the overall obviousness analysis (i.e. as opposed to analyzing the prior art, reaching a tentative conclusion, and then assessing whether objective indicia alter that conclusion).

(15.) Put another way, my understanding is that not all innovations are patentable. Even if a claimed product or method is not disclosed in its entirety in a single prior art reference, the patent claim is invalid if the invention would have been obvious to a person of ordinary skill in the art at the time of the invention. In particular, I understand that a patent claim is normally invalid if it would have been

a matter of “ordinary innovation” within the relevant field to create the claimed product at the time of the invention.

(16.) In determining whether the subject matter as a whole would have been obvious at the time that the invention was made to a person having ordinary skill in the art, I have been informed that rationales that may support a conclusion of obviousness include:

- a. Combining prior art elements according to known methods to yield predictable results;
- b. Simple substitution of one known element for another to obtain predictable results;
- c. Use of known technique(s) to improve similar methods or apparatuses in the same way;
- d. Applying a known technique to a known method or apparatus ready for improvement to yield predictable results;
- e. “Obvious to try”—choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;
- f. Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to one of ordinary skill in the art;
- g. Some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

(17.) The “teaching, suggestion, or motivation” test is a useful guide in establishing a rationale for combining elements of the prior art. This test poses the

question as to whether there is an explicit teaching, suggestion, or motivation in the prior art to combine prior art elements in a way that realizes the claimed invention. Though useful to the obviousness inquiry, I understand that this test should not be treated as a rigid rule. It is not necessary to seek out precise teachings; it is permissible to consider the inferences and creative steps that a person of ordinary skill in the art (who is considered to have an ordinary level of creativity and is not an “automaton”) would employ.

II. Description of the Relevant Field and the Relevant Timeframe

(18.) I have carefully reviewed the ‘344 Patent as well as the file history of the ‘344 Patent. Based on my review of this material, I believe that the relevant field for the purposes of the ‘344 Patent is generally computer security. In particular, the ‘344 Patent is related to anti-virus computer software that can dynamically detect viruses and other malware by examining certain characteristics of the software being examined. I have been informed that the relevant timeframe is on or before June 30, 2006.

(19.) As described in Section I above and as shown in my CV, I have extensive experience in the field of computer engineering, computer programming and security, and anti-virus computer software. Based on my experience, I have a good understanding of the relevant field in the relevant timeframe.

III. The Person of Ordinary Skill in the Relevant Field in the Relevant Timeframe

(20.) I have been asked to provide my opinion regarding the qualifications of a person having ordinary skill in the art (“POSITA”) with respect to the ‘344 Patent. I understand that the POSITA is a hypothetical person who is presumed to have known the relevant art at the time of the invention, which I understand here is on or before June 30, 2006. I understand that factors that may be considered in determining the level of ordinary skill in the art may include: (a) the type of problems encountered in the art; (b) prior art solutions to those problems; (c) the rapidity with which innovations are made; (d) the sophistication of the technology; and (e) the educational level of active workers in the field.

(21.) In my opinion, the POSITA relevant to the ‘344 Patent, at the time the ‘344 Patent was effectively filed, would have at least 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.

(22.) Based on my experience, I have an understanding of the capabilities of a person of ordinary skill in the relevant field. I have supervised and directed many such persons over the course of my career. Further, I had those capabilities myself at the time the patent was effectively filed.

IV. Engineering Background Underlying the ‘344 Patent

(23.) I will sometimes refer to the state of the art as “before the ‘344 Patent.” I have been informed that the relevant timeframe is on or before June 30, 2006. Accordingly, when I speak about the state of the art “before the ‘344 Patent,” I mean at least before June 30, 2006.

(24.) A computer program is a sequence of instructions that tell a computer processor what to do. Although the author of a computer program would be understandably upset if processors refused to obey the instructions, the blind obedience they are designed to deliver makes them incredibly vulnerable to misuse. Computer processors will not question the instructions they are given no matter how harmful. And processors can run millions and billions of instructions per second all day and all night whether a human is monitoring them or not.

(25.) Even when they can be monitored, computer instructions in a software program are generally too large, too complicated, and too cryptic for a human to easily review. Even if they could be inspected, it is very difficult to ensure that they are not modified at any time before they reach the processor.

(26.) Perhaps an even greater complication is that computer scientists have proven that it is impossible to find a general solution to this problem. This means that there will never be a perfect method for detecting any and all harmful instructions. *See, e.g.,* Ex. 1012 and 1013.

(27.) Accordingly, those individuals and organizations that attempt to produce harmful computer instructions and those that attempt to stop them have been and will always be in an arms race. Each year, the “bad guys” will find a new way to get processors to do things they should not and each year the “good guys” will have to find a new ways to defeat them. This ongoing struggle was already in full swing by the beginning of the 1990’s as the anti-virus community matured both in academic research and commercial products.¹

(28.) One of the earliest approaches to defeating viruses and other kinds of malware is still used today: scanners.

(29.) The basic concept behind a scanner is to search for known patterns and/or sequences of malware in computer resources. Ex. 1014 at 7. Viruses work by inserting instructions that hijack execution of normal programs. When the normal program is executing the inserted viral code takes over. It generally performs some kind of replication of itself, inserting the viral code into other programs, along with other functions. *See, e.g.*, Ex. 1012 and 1015.

¹ As a side note about nomenclature, any harmful computer code is generically called malware. Within that broad category are subtypes such as computer virus, Trojan horse, worm, and so forth. Nevertheless, these distinctions are generally related to how the malware “broke in” or how it spread. These kinds of issues are the purview of the security practitioner. The consumer, for the most part, does not care how the bad guy deleted their photos of their kids or how it got to their computer. For these reasons, the most common and most recognize type of malware, a virus, is often used as a placeholder for malware of any kind.

(30.) Early scanners emerged in 1988 and as late as 1991 most viruses could be detected by searching for an exact sequence of bytes. Each of these kinds of virus has a specific sequence of code that could, with relative ease, be recognized. Ex. 1014 at 7 and Ex. 1016 at 7. Nevertheless, antivirus researchers already could see the next generation of viruses on the horizon that would be harder to detect, either because the virus is obscured or encrypted, or because the virus can have multiple forms. Ex. 1016 at 19 and Ex. 1017 at 15-16.

(31.) Antivirus researchers were already working on more advanced signature matching in the 1991 timeframe. One approach used by a number of commercial and academic parties was a “signature” that was actually a type of program or script that could be executed by the antivirus software. These signatures contained instructions about where to look for virus code in files, mechanisms for removing encryption or garbling, and even instructions for repair. In this way, individual “signatures” became more advanced search-and-destroy tools. These types of signatures were generically said to be written in a “virus description language.”

Virus-description languages for this, and other purposes, have been around for some time; Christoph Fischer at the University of Karlsruhe, Morton Swimmer at the University of Hamburg, Alan Solomon in the UK and no doubt many others in the field have worked on similar things. Ex. 1016 at 8.

(32.) By 1993, signatures advanced again, largely to deal with the increasing sophistication of “polymorphic” viruses. These viruses could change shape or form in a wide range of ways. New signatures not only understood viruses, they understood structures and flow of normal programs. The polymorphic viruses of this time frame could be found by tracing the execution path of the program to find decryption operations. Some antivirus products used a common decryption engine and then looked for signatures of the decrypted code. But as polymorphic viruses became more specialized, individual signatures had to be tailored to individual polymorphic encryption routines. Ex. 1018 at 13-15.

(33.) For much of the history of signatures, researchers recognized that the true weakness of a signature is its inability to recognize completely new types of threats. Only after a new virus emerged, potentially doing significant damage, could a signature be written and deployed. Accordingly, many antivirus products also used a mechanism called “heuristic scanning” to observe “virus like behavior” in programs. This way, the antivirus products would look for functionalities of the software to determine whether if it is malware. In other words, the antivirus products would look at what the software is doing. This idea was well known. For example, in 1995, one researcher suggested that heuristic scanning was a necessary part of the antivirus future. *See* Ex. 1019.

(34.) This researcher, Dmitry Gryaznov, suggested that traditional approaches to scanning were already running into serious difficulty.

As mentioned above, the main drawback of scanners is that they can detect only known computer viruses. Six or seven years ago, this was not a big deal. New viruses appeared rarely... There were about 3,000 known DOS viruses at the beginning of 1994. A year later, in January 1995, the number of viruses was estimated at least 5,000. Another six months later, in July 1995, the number exceeded 7,000. Many anti-virus experts expect the number of known DOS viruses to reach the 10,000 mark by the end of 1995... Even monthly scanner updates are often late, by one month at least! ... Just [a] few years ago a 360Kb floppy diskette would be enough to hold half a dozen popular scanners, leaving plenty of room for system files to make the diskette bootable. Today, an average good signature-based scanner alone would occupy at least a 720Kb floppy, leaving virtually no room for anything else. Ex. 1019 at 228.

(35.) Gryaznov noted that anti-virus researchers could often glance at the code of a program and instantly recognize if it had a virus, even one that had not been previously seen. He went on to explain that many viruses of the day all had some common operations. He summarized one set of these operations this way:

1. The program immediately passes control close to the end of itself.
2. It modifies some bytes at the beginning of its copy in memory.
3. Then it starts looking for executable files on a disk.
4. When found, a file is opened.
5. Some data is read from the file.
6. Some data is written to the end of the file. Ex. 1019 at 229.

(36.) By looking for files with these kinds of operations, one can categorize a range of viruses even if they are entirely new. On the other hand, Gryaznov also

observed that some legitimate programs have these same features. One highly ironic example is virus scanners themselves. Ex. 1019 at 230-31.

(37.) In order to avoid false positives, Gryaznov also emphasized the need for “negative heuristics,” or signs of a legitimate program. By combining the positive heuristics and negative heuristics Gryaznov was optimistic that even new viruses could be detected much of the time with very few false positives. Ex. 1019 at 231.

(38.) Another significant change that happened in the 1995 time frame was that viruses moved out of modifying binary code and into the world of scripts. A script is a computer program which is not executed directly. Instead, it is “interpreted” by another computer program. Common today, scripts only began to be popular in the 1995 time frame in the form of Microsoft Word macros. And, unsurprisingly, a new type of virus that could infect these scripts emerged by the end of that year. Ex. 1020 at 8-9. The concept, however, had been known to researchers since at least 1990. Ex. 1021 at 11-12.

(39.) Because a virus must be executed to take effect, for year the antivirus scanners only had to scan executable files. The introduction of Macro viruses that could be contained in office documents required that the number of files scanned to be expanded significantly. Ex. 1020 at 8-9. Moreover, all of the heuristics that worked for executable files had to be updated for these new strains of malware.

(40.) By the year 2000, macro viruses were the most significant problem for the antivirus community. Scanners and heuristics were, again, the tools most commonly used to combat them. Ex. 1022 at 10-11.

(41.) By 2006, the time of the ‘344 Patent’s filing, scanners had been around for eighteen years. During those nearly two-decades of development, they have been investigated, explored, revised, and updated countless times. Vast libraries of advanced signatures have been assembled for academic research and commercial enterprises. These advanced signatures include heuristics that record the functionalities or behaviors of malware. Anti-virus software that scan for generic “bad” behavior have been used since the beginning with various rates of success and with various rates of false positives. Even today, scanners using libraries of advanced signatures, heuristics, and other records of functionalities or behaviors are the basis of most anti-virus products.

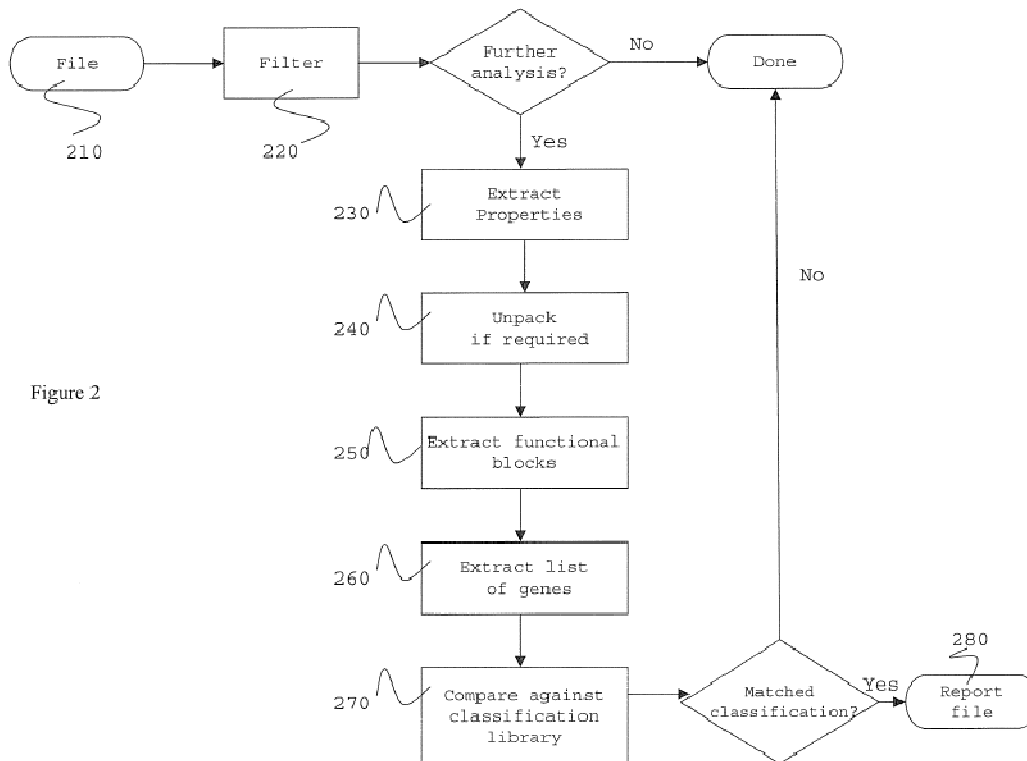
V. The ‘344 Patent

(42.) The ‘344 Patent claims methods for detecting malware by identifying “genes” from the suspected malware and matching the identified genes against a library containing classifications based on groupings of genes. Ex. 1001 at claim 1. “Genes” is not a term of art ordinarily used in computer science. Rather, it generally is a term used in the biological sciences. Although the patentee coined a new term in computer science by repurposing a biological term, the concepts that

relate to the term “genes,” as explained in the ‘344 Patent, are not new. As I explained above and in detail below, virus or malware detection by dynamically identifying features of the software and comparing the identified features to a library of known malware features was well-known in the prior art, and one of ordinary skill in the art would have combined any of the well-known design choices/features of these systems to arrive at the claimed invention. Calling these identified features “genes” does not render the ‘344 Patent novel. My opinion is consistent with the construction I was asked to apply for this Declaration, which construes “gene” generically as “a piece of functionality or property of a program.” The ‘344 Patent offers nothing new, novel, or inventive that was not already known and widely practiced at the time the patent was originally filed.

(43.) As shown in FIG. 2 (below), the ‘344 Patent describes a malware detection system. The operation of the system is as follows. The system first identifies one or more “functional blocks” from the examined software. Ex. 1001 at 5:17-28. The ‘344 Patent explains that these “functional blocks” are “sequences of application program interface (API) calls, string references, etc., which illustrate the function and execution flow of the program.” *Id.* The ‘344 Patent then describes that one or more “genes” are identified from these functional blocks. Ex. 1001 at 7:64-67. The ‘344 Patent explains that “[a] gene is a piece of functionality or property of a program” and that “[e]ach piece of functionality is described using

sequences of APIs and strings.” *Id.* The ‘344 Patent then describes that the “genes” are matched 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.* If there is a match, the software is identified as malware and it may be removed or blocked. Ex. 1001 at 4:44-47.



(44.) Although not explained in detail in the specification, claim 16 of the ‘344 Patent also claims a method for generating classifications for the library. Ex. 1001 at claim 16. Claim 16 recites testing the set of genes for false-positives against one or more reference files using a processor; defining the software

classification based on the set of genes; and storing the set of genes and the software classification in the library. Ex. 1001 at 8:49-54.

VI. Claim Interpretation

(45.) In proceedings before the USPTO, I understand that the claims of an unexpired patent are to be given their broadest reasonable interpretation (“BRI”) in view of the specification from the perspective of one skilled in the art.

(46.) I have been provided and reviewed the proposed claim constructions offered by the Patent Owner, *Fortinet, Inc. v. Sophos, Inc.*, Case No. 3:13-cv-05831-EMC (N.D. Cal.) (Ex. 1008). I have also been provided and reviewed Patent Owner’s infringement contentions in *Fortinet, Inc. v. Sophos, Inc.*, Case No. 3:13-cv-05831-EMC (N.D. Cal.) (Ex. 1009.)

(47.) I have been asked to assume that the Patent Owner considers the claim constructions set forth in those materials to reflect the Patent Owner’s appropriate interpretation of the claim terms in the District Court proceeding, and that such a construction would be within the scope of the broadest reasonable interpretation of the claim terms. I therefore rely on Patent Owner’s proposed constructions and infringement contentions in the District Court as evidence of the Patent Owner’s understanding of the BRI for the claims of the ‘344 Patent.

(48.) I also have been informed that challenges based on enablement, indefiniteness, or written description under U.S. C. § 112 are not available in *inter*

partes review proceedings. I have therefore have not considered whether the claims terms are enabled, indefinite, or supported by written description for the purposes of this declaration.

(49.) Accordingly, for purposes of this declaration, I have construed the claim limitations as follows:

(50.) “**gene**”— *a piece of functionality or property of a program.*

(51.) “**functional block**”— *a segment of the program that illustrates the function and execution flow of the program.*

VII. Discussion of Relevant Patents and Publications

(52.) As noted above, I have been asked to consider the teachings of the prior art cited in the concurrently filed petition in view of the knowledge held by one of ordinary skill and whether the skilled practitioner would have combined the references as applied in the petition.

A. Unpatentability in view of Bodorin

3. Claims 1 and 2 are anticipated by Bodorin

(53.) U.S. Patent No. 7,913,305 to Bodorin et al. (“Bodorin”) was filed on January 30, 2004. 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:

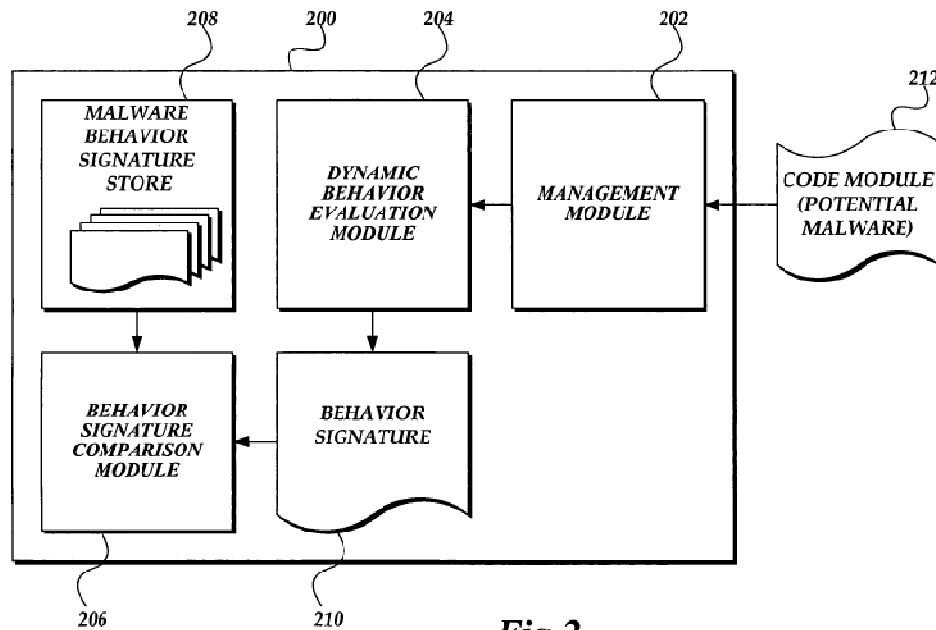


Fig.2.

Ex. 1003 at Fig. 2.

(54.) The management module 202 first evaluates the code module 212 to determine the appropriate type of dynamic behavior evaluation module needed. Ex. 1003 at 4:26-29. Once determined, the code module 212 is run 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. Interesting behaviors include:

TABLE A

RegisterServiceProcess ()	ExitProcess ()
Sleep ()	GetCommandLine ()
WinExec (application__name)	CreateProcessA (proces__name, parameters__list)
InternetOpenUrlA (URL__name)	GlobalAlloc ()
InternetOpenA (URL__name)	GetTickCount ()
IntenetCloseHandle ()	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__specifcator)
RegSetValuExA (subkey__name)	FindNextFileA ()
RegCloseKey ()	FindClose ()
UrlDownloadToFileA (url__name, file__name)	

Ex. 1003 at 5:2-20.

(55.) As shown in the table, the interesting behaviors include API calls such as RegisterServiceProcess() and strings such as application_name. Bodorin collects recorded interesting behaviors into behavior signatures and they are illustrated in Figs. 5A and 5B:

500	502	504	506	508
INTERNET_OPEN_URL	MATCH_ANYTHING	NULL	NULL	
INTERNET_READ_FILE	1	NULL	NULL	
WRITE_FILE	NULL	NULL	NULL	
OPEN_REGISTRY_KEY	"OPERATINGSYSTEM\\CURRENTVERSION\\RUN"	1	NULL	
SET_REGISTRY_VALUE	MATCH_ANYTHING	NULL	NULL	
EXECUTE	MATCH_FILENAME".EXE"	1	NULL	

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.

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

(56.) Once the behavior signatures have 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.

(57.) In connection with the above, I have reviewed, had input into, and endorse as if set forth fully herein the claim charts in the accompanying petition showing that each element of claims 1 and 2 are disclosed by Bodorin.

(58.) Specifically first, like the ‘344 Patent, Bodorin discloses a method for classifying software. Bodorin discloses malware detection system that determines whether an executable code module is malware according to behaviors exhibited while executing. Ex. 1003 at Abstract. The detection of malware is classifying software.

(59.) Next, Bodorin discloses providing a library of gene information including a number of classifications based on groupings of genes. Bodorin discloses extracting “interesting behaviors” from software, examples of which are listed in Table A.

TABLE A

RegisterServiceProcess ()	ExitProcess ()
Sleep ()	GetCommandLine ()
WinExec (application__name)	CreateProcessA (proces__name, parameters__list)
InternetOpenUrlA (URL__name)	GlobalAlloc ()
InternetOpenA (URL__name)	GetTickCount ()
IntenetCloseHandle ()	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__specifier)
RegSetValuExA (subkey__name)	FindNextFileA ()
RegCloseKey ()	FindClose ()
UrlDownloadToFileA (url__name, file__name)	

Ex. 1003 at 5:2-20.

(60.) As shown in the table, the interesting behaviors are based on functionality in the form of API calls and strings such as URL names. These “interesting behaviors” map to “genes,” construed as “a piece of functionality or property of a program.” For reference, the specification of the ‘344 Patent gives the following as examples of APIs used in gene definitions: GetSystemDirectoryA, GetModuleFileNameA, RegSetValueExA, CopyFileA, and CreateProcessA. Ex. 1001 at 6:11-45. These API calls are standard Windows API calls and both Bodorin and the ‘344 Patent use them in the same way. It should be noted that a single Bodorin behavior may combine both an API call and one or more strings that represent parameters. Ex. 1003 at 4:58-61.

(61.) These behaviors are compiled into a “behavior signatures,” as shown in Figs. 5A and 5B of Bodorin, which map to the “groupings of genes” as claimed. Bodorin discloses a library called the malware behavior signature store that is a repository of behavior signatures, which maps to the “library” claimed in the ‘344 Patent. Ex. 1003 at 5:27-43. The malware behavior signature store classifies the behavior signatures according to whether they are based on known malware. *Id.*

(62.) Next, Bodorin discloses identifying at least one functional block and at least one property of the software. As construed, “functional block” means a segment of the program that illustrates the function and execution flow of the program. As explained above, Bodorin discloses examining the suspected program

for behaviors. Ex. 1003 at 4:51-54. These behaviors include API calls that illustrate the function and execution flow of the program. A subset of these behaviors are extracted as being “interesting.”

(63.) Next, Bodorin discloses 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 explained above, Bodorin discloses the extraction of “interesting behaviors” that are based on APIs and strings. These “interesting behaviors” are “genes,” construed as “a piece of functionality or property of a program,” that describe 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.

(64.) Next, Bodorin discloses matching the one or more genes against one or more of the number of classifications using a processor and classifying the software based on the matching to provide a classification for the software. Ex. 1003 at 5:27-43. If the extracted behavior signature is matched in the store, then the software is classified as malware. Ex. 1003 at 5:44-49.

(65.) Finally, Bodorin discloses notifying a user of the classification of the software. Ex. 1003 at 5:44-49. Accordingly, Claims 1 and 2 are anticipated by Bodorin.

4. Claims 10 and 13 are obvious in light of Bodorin itself

(66.) 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.

(67.) Since the creation of signature scanners more than a decade and a half before the '344 Patent, the stated advantage of a scanner over a behavior blocker or other defensive mechanism was that the scanner could proactively prevent damage. Consider this discussion of scanners from 1995:

More generic anti-virus tools [than scanners]... can detect a virus... only **after** the virus has run an infected other programs. In many cases, the risk of allowing a virus to run on your computer is just not affordable. Using a heuristic scanner... allows detection of most new viruses... **before** an infected program is copied to your system and executed. Ex. 1019 at 227 (emphasis in original).

(68.) Those of skill in the art would recognize that there would be little advantage to Bodorin if it did not proactively remove and/or block access to identified malware.

(69.) Unsurprisingly then, there is support 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

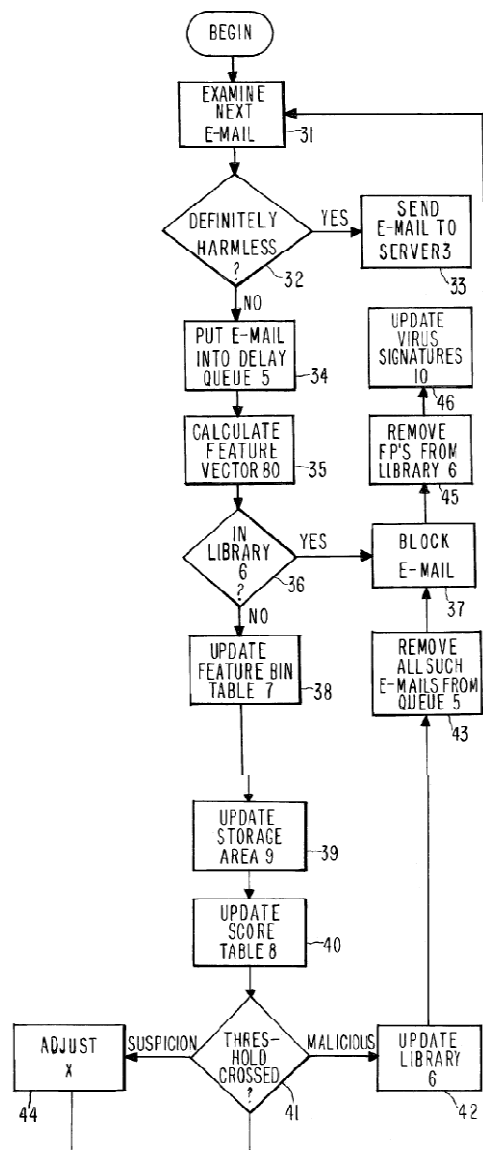
known malware. Ex. 1003 at 1:44-61. Bodorin then explicitly states that it would be advantageous to combine the dynamic malware detection system it discloses with prior art anti-virus software. Ex. 1003 at 3:58-4:12. Specifically, Bodorin states “while the malware detection system may be used as a stand-alone 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 that removes and blocks access to identified malware.

(70.) In connection with the above, I have reviewed, had input into, and endorse as if set forth fully herein the claim charts in the accompanying petition showing that each element of claims 10 and 13 are obvious in light of Bodorin itself to a person of ordinary skill in the field. Accordingly, Claims 10 and 13 are obvious in light of Bodorin itself.

B. Unpatentability in view of Kissel

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

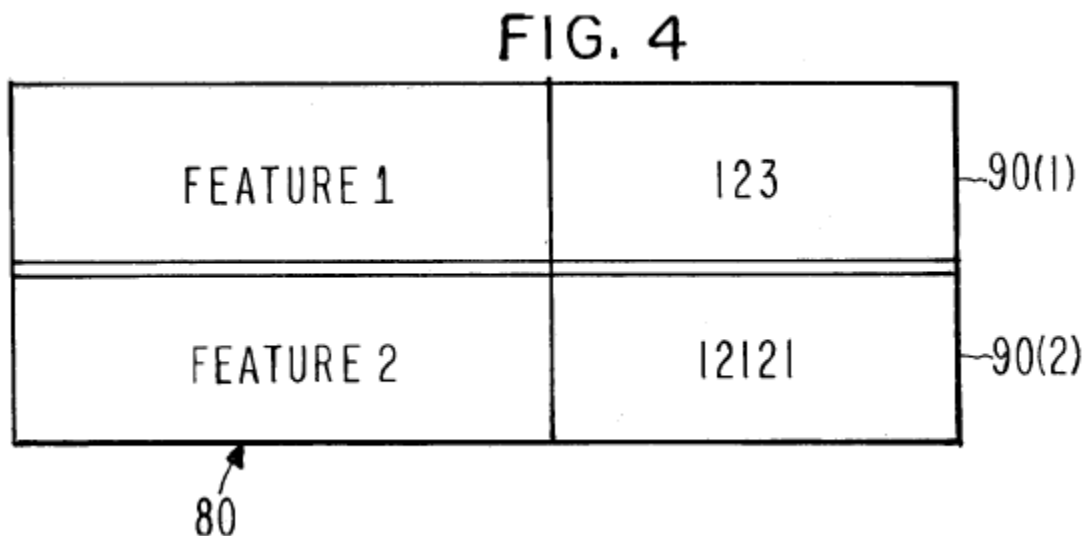
(71.) U.S. Patent No. 7,373,664 to Kissel (“Kissel”) was filed on December 16, 2002. Kissel discloses a system for detecting the presence of malicious computer code in emails. Ex. 1004 at 1:44-46. The operation of the system is generally illustrated by Figs. 3A and 3B:



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

(72.) As shown in the figure, if the email is not deemed to be definitely harmless, *e.g.*, it has no attachments or script content, 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, the features extracted may also include a tokenized representation of a script program within an email, which may include JavaScript API calls and strings. Keywords, for example “var” and “if,” are searched for within this set of APIs and strings. Ex. 1004 at 3:45-4:4.

(73.) The extract feature vector is shown in Fig. 4.



Ex. 1004 at Figs. 4.

(74.) This feature vector is then examined against the blocked feature instance library in step 36. The library is shown in Fig. 5.

FIG. 5

FEATURE 1	789	90(3)
FEATURE 2	8133457	90(4)

Ex. 1004 at Figs. 5.

(75.) 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.

(76.) In connection with the above, I have reviewed, had input into, and endorse as if set forth fully herein the claim charts in the accompanying petition showing that each element of claims 1, 2, 10, and 13 are disclosed by Kissel.

(77.) Specifically first, like the '344 Patent, Kissel discloses a method for classifying software. Kissel discloses methods, apparatus, and computer-readable media for detecting the presence of malicious computer code in a plurality of e-

mails. Ex. 1004 at Abstract. The detection of malicious computer code is classifying software.

(78.) Next, Kissel discloses providing a library of gene information including a number of classifications based on groupings of genes. Kissel discloses extracting “features” from software, which may include the name of each e-mail attachment, the size of each e-mail attachment, and keywords from parsed scripts. Ex. 1004 at 3:35-4:4. These features map to “genes,” construed as “a piece of functionality or property of a program.” These features are compiled into a “feature vector,” as shown in Fig. 4 of Kissel, which map to the “groupings of genes” in the ‘344 Patent. These features vectors in turn are stored in a blocked feature instance library, which maps to the “library” claimed in the ‘344 Patent. Ex. 1004 at 3:23-30. The blocked feature instance library classifies the feature vectors according to whether they are indicative of malicious code. *Id.*

(79.) Next, Kissel discloses identifying at least one functional block and at least one property of the software. As construed, “functional block” means a segment of the program that illustrates the function and execution flow of the program. Kissel discloses that in extracting the “features,” its system identifies segments of the program that illustrates the function and execution flow of the program, such as parsed scripts. Ex. 1004 at 3:35-46. Kissel also discloses that its system identifies script programs within an email body. This identification process

includes extracting a script from within <SCRIPT> tags and analyzing the extracted script using parser or interpreter. The parsing process produces a tokenized representation of the program where individual tokens and groups of tokens represent functions and execution flow of the program. Ex. 1004 at 3:43-61. The name of the attachment would be a string, and the parsed script, which Kissel discloses is stored as a tokenized representation, would be a sequence of APIs.

(80.) Next, Kissel discloses 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 explained above, Kissel discloses the extraction of “features” from software, which may include the name of each e-mail attachment, the size of each e-mail attachment, and keywords from parsed scripts. Ex. 1004 at 3:35-46. As explained above, Kissel discloses parsing scripts to produce a tokenized representation. Kissel then discloses looking for keywords in the tokenized representation, and store the extracted keywords as “features.” Ex. 1004 at 3:64-4:5. These “features” maps to “genes,” as construed, that describe 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.

(81.) Next, Kissel discloses matching the one or more genes against one or more of the number of classifications using a processor and classifying the

software based on the matching to provide a classification for the software. This occurs at step 36 in Fig. 3A. If the extracted feature is matched in the library, then the software is classified as malware.

(82.) Finally, Kissel discloses notifying a user of the classification of the software, removing, or blocking access to the software. Kissel discloses that once the software is identified as malware, it is “blocked.” Ex. 1004 at 5:47-51. In the context of Kissel, “blocked” can be notifying a user, removing, or blocking access to the software. Ex. 1004 at 5:55-58. Accordingly, Claims 1, 2, 10, and 13 are anticipated by Kissel.

2. Claims 16 and 17 are obvious in light of Kissel in combination with Apap

(83.) As shown in steps 36-42 in Figs. 3A and 3B of Kissel and explained in column 6, lines 7 through 38, if the extracted feature vector is not in library, it is stored and a score associated with the extracted feature vector is generated. If the score exceeds a certain threshold, it indicates that this unknown extracted feature vector is associated with malware and the library is updated. Ex. 1004 at 6:7-38. Kissel further discloses calibrating the threshold to test whether these new extracted feature vectors are false-positives, i.e. not associated with malware. Ex. 1004 at 8:47-62.

(84.) U.S. Patent No. 7,448,084 to Apap et al. (“Apap”) was filed on January 27, 2003. 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.

(85.) 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.

(86.) 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. 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.

(87.) Even from the earliest years of anti-virus research and commercial operations, the testing of scanners and other tools against reference files was not just common but promoted. As far back as 1991, “Virus Bulletin,” published an anti-virus signature that ended up falsely identifying key aspects of the operating system as infected. The following month, “Virus Bulletin” published a guest editorial entitled “The Mother of All False Positives.” This editorial promoted a number of methods for improving the accuracy of scanners. Specifically, “[s]ignatures should be carefully tested before they are released, on as large a corpus of uninfected programs as possible, to help reduce the incidence of obvious false positives.” Ex. 1017 at 2.

(88.) The fact that this method was well known is also demonstrated not only by Apap itself and public statements from practitioners of the art, 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.

(89.) 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. 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.

(90.) In connection with the above, I have reviewed, had input into, and endorse as if set forth fully herein the claim charts in the accompanying petition showing that each element of claims 16 and 17 are disclosed by Kissel and Apap, and that, as set forth above, a person of ordinary skill in the field would have combined the teachings of the references as claimed. Accordingly, Claims 16 and 17 are obvious in light of Kissel in combination with Apap.

C. Unpatentability in view of Bodorin and Kissel, further in view of Apap

1. Claims 16 and 17 are obvious in light of Bodorin in combination with Kissel and Apap

(91.) As explained above, Bodorin discloses a malware detection system that generates behavior signature from programs, which corresponds to the "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 generated behavior signature, which is claimed in claim 16. However, as explained in detail above, Kissel in combination with Apap disclose every limitation of claim 16.

(92.) 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.

(93.) Furthermore, there was explicit teaching, suggestion, and motivation generally in the art to combine. New malware appear regularly and frequently. 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. Accordingly, one of ordinary skill in the art would be motivated to combine Bodorin with Kissel and Apap.

(94.) In connection with the above, I have reviewed, had input into, and endorse as if set forth fully herein the claim charts in the accompanying petition showing that each element of claims 16 and 17 are disclosed by Bodorin and Apap, and that, as set forth above, a person of ordinary skill in the field would have combined the teachings of the references as claimed. Accordingly, Claims 16 and 17 are obvious in light of Bodorin in combination with Kissel and Apap.

D. Unpatentability in view of Chen

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

(95.) U.S. Patent No. 5,591,698 to Chen et al. (“Chen”) issued on September 1999. 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.

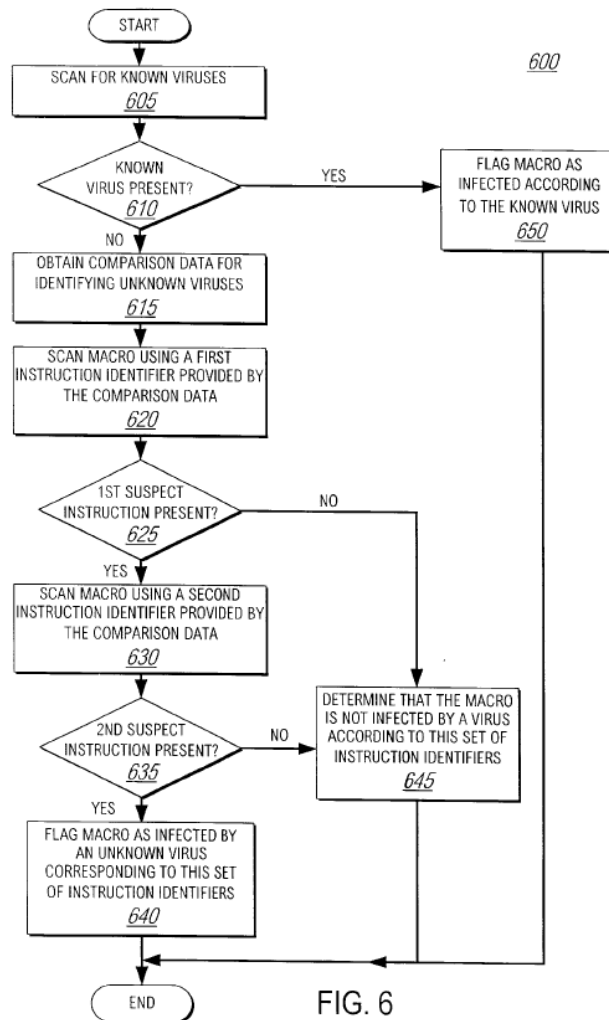


FIG. 6

Ex. 1005 at Fig. 6.

(96.) 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 instructions 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.

900

903 SET #	904 INSTRUCTION ID#	905 INSTRUCTION IDENTIFIER (TEXT/HEX)
902 1	1	.Format = 1 73 CB 00 0C 6C 01 00
	2	Macro Copy 67 C2 80
902 2	1	.Format = 1 73 CB 00 0C 6C 01 00
	2	Organizer.Copy 64 6F 02 67 DE 00 73 87 02 12 73 7F
3	1	.Format = 1 73 CB 00 0C 6C 01 00
	2	macros. 6D 61 63 72 6F 73 76 08
4	1	FileSaveAs a\$,1 12 6C 01 00
	2	MacroCopy 64 67 C2 80 6A 0F 47
5	1	ylformat c: /u" 79 7C 66 6F 72 6D 61 74 20 63 6A
	2	Environ\$ ("COMSPEC") 80 05 6A 07 043 4F 4D
.	.	.
.	.	.
.	.	.
902 i	1	...
	2	...

	j	...

FIG. 9

Ex. 1005 at Fig. 9.

(97.) 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.

(98.) In connection with the above, I have reviewed, had input into, and endorse as if set forth fully herein the claim charts in the accompanying petition showing that each element of claims 1, 2, 10, and 13 are disclosed by Chen.

(99.) Specifically, first, like the '344 Patent, Chen discloses a method for classifying software. Chen discloses a method to detect and remove viruses from macros. Ex. 1005 at Abstract. The detection and removal of viruses from macros is classifying software.

(100.) Next, Chen discloses providing a library of gene information including a number of classifications based on groupings of genes. Chen discloses a data table including exemplary sets of instruction identifiers which are stored in the virus information module. Ex. 1005 at 14:65-15:13. As shown in Fig. 9 of Chen, this “library” (*i.e.* the virus information module) includes “a number of classifications” (*i.e.* a number of sets of instructions, as shown in column 903) “based on groupings of genes” (*i.e.* groups of instruction identifiers for each set of instructions, as shown in column 905).

900

903 SET #	904 INSTRUCTION ID#	905 INSTRUCTION IDENTIFIER (TEXT/HEX)
902 1	1	.Format = 1 73 CB 00 0C 6C 01 00
	2	Macro Copy 67 C2 80
902 2	1	.Format = 1 73 CB 00 0C 6C 01 00
	2	Organizer .Copy 64 6F 02 67 DE 00 73 87 02 12 73 7F
3	1	.Format = 1 73 CB 00 0C 6C 01 00
	2	macros. 6D 61 63 72 6F 73 76 08
4	1	FileSaveAs a\$,1 12 6C 01 00
	2	MacroCopy 64 67 C2 80 6A 0F 47
5	1	ylformat c: /u" 79 7C 66 6F 72 6D 61 74 20 63 6A
	2	Environ\$ ("COMSPEC") 80 05 6A 07 043 4F 4D
.	.	.
.	.	.
.	.	.
902 i	1	...
	2	...

	j	...

FIG. 9

Ex. 1005 at Fig. 9.

(101.) Next, Chen discloses identifying at least one functional block and at least one property of the software. As construed, “functional block” means a segment of the program that illustrates the function and execution flow of the program. Chen discloses that macros are scanned for suspect “instructions,” which are segments of the macro that illustrate the function and execution flow of the program. For example, Chen discloses that conventional macros include various instructions including those for relatively simple operations such as keystrokes as well as those for operations such as opening, copying and deleting files. Ex. 1005

at 5:16-37. Identifying these instructions also identify at least one property of the macro. For example, identifying a copy command in the macro would also identify that a property of the macro is to copy a certain file. As shown in Fig. 9, these suspect instructions 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.

(102.) Next, Chen discloses 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, matching the one or more genes against one or more of the number of classifications using a processor and classifying the software based on the matching to provide a classification for the software. As explained in the above paragraph, Chen discloses identifying functional blocks or properties of the software as a sequence of APIs and strings. Chen also discloses that the virus scanning module of the system obtains a set of instructions identifiers from the virus information module. Ex. 1005 at 13:20-33. As explained above, the instruction identifiers map to the “genes” recited in the claims. The virus scanning module then scans the macro and identifies the instructions in the macro when there is a match. Ex. 1005 at 15:13-31. When two or more instructions are matched, the system disclosed in Chen classifies the macro as being infected. Ex. 1005 at 15:43-54.

(103.) Finally, Chen discloses notifying a user of the classification of the software, removing, or blocking access to the software. Chen discloses that once the software is identified as being infected by a virus, various corrective action can be taken, including flagging the macro as being infected or removing the virus so as to block access to the virus. Ex. 1005 at 15:43-54; 16:24-47. Accordingly, Claims 1, 2, 10, and 13 are anticipated by Chen.

VIII. Secondary Considerations

(104.) I am unaware of any objective indicia or so-called “secondary considerations” that would support a finding of non-obviousness of the ‘344 Patent.

(105.) Prior to being involved in *Fortinet, Inc. v. Sophos, Inc.*, Case No. 3:13-cv-05831-EMC (N.D. Cal.), I had never heard of the ‘344 Patent or its inventors. I am unaware of any praise for the ‘344 Patent, or of any commercial success attributable to the alleged inventions claimed in the ‘344 Patent.

(106.) In my opinion, there was not a long felt but unresolved need at the time of the invention for the alleged inventions claimed in the ‘344 Patent. In my opinion, the purported problem and solution addressed in the ‘344 Patent were identified and solved long before the filing date of the ‘344 Patent, as I discussed in the Background section, and demonstrated by systems such as those disclosed in Bodorin and Kissel, as discussed above.

(107.) I am unaware of anyone expressing skepticism about the alleged inventions claimed in the '344 Patent. As addressed above, the subject matter of the claims was well known in the field of computer security long before the filing date of the '344 Patent, and persons of ordinary skill in the art would have considered it routine to implement the claimed subject matter.

(108.) In my opinion, the inventors of the '344 Patent did not achieve any surprising or unexpected results. As detailed above, the claimed invention was not only known in the art long before the filing date of the '344 Patent, but also was nothing more than a predictable combination of known prior art element.

(109.) I am also unaware of any copying of the alleged inventions claimed in the '344 Patent.

(110.) Thus, I am unaware of any secondary considerations supporting the non-obviousness of the '344 Patent. To the extent Patent Owner responds with evidence purporting to show objective indicia of non-obviousness, I reserve the right to address and respond to the evidence in a supplemental declaration.

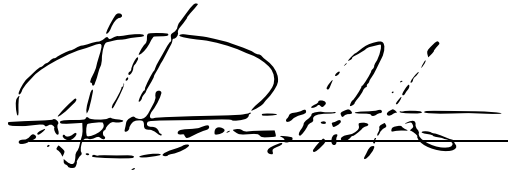
IV. Additional Statements

For my efforts in connection with the preparation of this declaration I have been compensated at my standard hourly rate for this type of consulting activity. My compensation is in no way contingent on the results of these or any other proceedings relating to the above-captioned patent.

In signing this declaration, I understand that the declaration will be filed as evidence in a review proceeding before the Patent Trial and Appeal Board of the U.S. Patent and Trademark Office. I acknowledge that I may be subject to cross examination in the case and that cross examination will take place within the United States. If cross examination is required of me, I will appear for cross examination within the United States during the time allotted for cross examination.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1101 of Title 18 of the United States Code and that such willful false statements may jeopardize the results of these proceedings.

Dated: January 23, 2015

A handwritten signature in black ink, appearing to read 'Seth Nielson', written over a horizontal line.

Seth Nielson, Ph.D.

APPENDIX A

Seth James Nielson, Ph.D.

410.497.7384

seth@harborlabs.com

Profile

I am a Principal at Harbor Labs with specialties in network security, network communications, software architecture, and programming languages. My objective is to enable clients to succeed in their technology projects by providing expertise, project management, and the capacity to make technological concepts accessible. I bring over a decade of industry and academic experience covering software application development, network protocols, cryptographic analysis, cryptographic algorithm implementation, reverse engineering, and vulnerability analysis. Drawing on these skills, I provide clients software reviews and analyses, security reviews, and technical insights into matters of intellectual property. I also provide technical support and expert witness services to litigation teams.

I completed my Ph.D. at Rice University in 2009 where my thesis investigated questions of security and anonymity in peer-to-peer (P2P) systems like BitTorrent. In addition to my professional work at Harbor Labs, I am an Adjunct Associate Research Scientist at Johns Hopkins University where I teach network security classes, mentor student capstone projects, and engage in academic research.

Education

<u>2009</u>	<i>Rice University</i>	Ph.D. in Computer Science
<u>2004</u>	<i>Brigham Young University</i>	M.S. in Computer Science
<u>2000</u>	<i>Brigham Young University</i>	B.S. in Computer Science

Academics and Research

<u>12/2014-Present</u>	<i>Johns Hopkins University</i>	Adjunct Associate Research Scientist
Teach graduate level courses on network security		
Advise student capstone projects		
Engage in academic research		
<u>1/2014-12/2014</u>	<i>Johns Hopkins University</i>	Lecturer
Teach graduate level courses on network security		
Advise student capstone projects		

Industry Experience

<u>2011-Present</u>	<i>Harbor Labs</i>	Research Scientist
<u>2005-2011</u>	<i>Independent Security Evaluators</i>	Senior Security Analyst
<u>2005</u>	<i>Google</i>	Summer Intern
<u>2001-2003</u>	<i>Metrowerks (Formerly Lineo, Inc.)</i>	Software Engineer II

Academic Awards

Brown Fellowship
John and Eileen Tietze Fellowship

Patents

Co-inventor: Orsini, R. 2014. Systems and methods for security data in motion. U.S. Patent 8,745,372 filed November 24, 2010 and issued June 3, 2014.

Co-inventor: Orsini, R. 2014. Systems and methods for security data in motion. U.S. Patent 8,745,379 filed August 20, 2012 and issued June 3, 2014.

Co-inventor: O'Hare, R. 2014. Systems and methods for security data. U.S. Patent 8,677,148 filed January 27, 2012 and issued March 18, 2014.

Publications

Aviel D. Rubin, Seth J. Nielson, Sam Small, Christopher K. Monson, *Guidelines for Source Code Review in Hi-Tech Litigation*, Harbor Labs White Paper (September 2013)

Seth James Nielson, *Reintroducing Pylogical*, BYU SEQuOIA Technical Report, (March 2012)

Seth James Nielson and Dan S. Wallach, *The BitTorrent Anonymity Marketplace*, arXiv Technical Report 1108.2718, (August 2011)

Seth James Nielson, Caleb E. Spare, and Dan S. Wallach, *Building Better Incentives for Robustness in BitTorrent*, arXiv Technical Report 1108.2716, (August 2011)

Seth James Nielson, *Designing Incentives for Peer-to-Peer Systems*, Rice University Department of Computer Science Ph.D. Thesis (2010)

Seth James Nielson and Charles D. Knutson, *Design Dysphasia and the Design Patterns Maintenance Cycle. Information & Software Technology*, volume 48, number 8, pp. 660- 675, (August 2006)

Seth James Nielson, Scott S. Crosby, and Dan S. Wallach, *A Taxonomy of Rational Attacks. In Proceedings of the Fourth International Workshop on Peer-to-Peer Systems (IPTPS '05)*, Ithica, New York, (February 2005)

Seth James Nielson, *OO++ Design Patterns, GOF Revisited*, Brigham Young University Department of Computer Science Master's Thesis (2004)

Seth James Nielson, Seth J. Fogarty, and Dan S. Wallach, *Attacks on Local Searching Tools*, arXiv Technical Report 1108.2704 (Originally produced in December, 2004, available on arXiv as of August 2011)

Rob Kunz, Seth Nielson, Mark Clement, Quinn Snell, *Effective Bandwidth for Traffic Engineering*, *Proceedings of the IEEE Workshop on High Performance Switching and Routing (HPSR 2001)*, Dallas, TX, (May 2001)

Special Projects

Creator and maintainer of a PROLOG-style logic programming module for Python available at <http://www.multiparadigm-python.org>.

Development of PLAYGROUND, a pedagogical model for network security instruction. **Public Release 2015.**

Technical Expertise

1/2001-Present

Languages:

Targets:

Software Development

C, C++, Java, Python, Objective-C, Assembly

Applications, libraries, device drivers, simulators, networking stacks, graphics, server code, security code, pedagogical tools, utilities, automation, GUIs, intrusion detection systems, attack simulation technology

<i>Toolkits:</i>	QT, Boost, Twisted, SWIG, test harnesses, CUDA
<i>Platforms:</i>	Windows, Linux, iOS
<u>9/2004-Present</u>	<i>Vulnerability and System Analysis</i>
<i>Examples:</i>	Google Desktop Search (2004), crypto protocols, viruses, malware, passwords, cryptographic implementation, security policy viability, marketplace viability and risks of existing and future products
<i>Tools:</i>	IDA Pro, port scanning, Formal cryptographic analysis tools, GCov and code coverage tools, fuzzing
<u>1/2010-Present</u>	<i>Source Code Review and Analysis</i>
<i>Samples:</i>	Antivirus software, firewall software, high-frequency trading algorithms, wireless protocol implementations, intrusion prevention software, email server software, document signature software
<i>Tools:</i>	Understand, customized scripts
<u>1/2010-Present</u>	<i>Technical Analysis of Intellectual Property</i>
<i>Issues:</i>	DMCA and copyright
<u>5/2010-Present</u>	<i>Technical Instruction</i>
	Teaching non-technical professionals about relevant high-tech operations
	Teaching technical professionals about technologies relevant to intellectual property
<u>5/2013-Present</u>	<i>Privacy Analysis</i>
<i>Projects:</i>	Review of corporate information gathering to assess privacy protections, Review of corporate information gathering to assist counsel in legal analysis
<u>9/2011-1/2012</u>	<i>Technical Project Management</i>
<i>Projects:</i>	Secure communication application, automated fuzzing tool
<i>Internal:</i>	Coding guidelines, manpower allocation, quality assurance
<i>Customer:</i>	Requirements analysis, budget and scheduling, conflict resolution
<u>9/2005-9/2011</u>	<i>Cryptographic Library Development</i>
<i>Algorithms:</i>	AES-GMAC, Shamir Key Splitting, Client-custom algorithms
<i>Special:</i>	GPU-accelerated AES (CUDA), file system integration, FIPS certified

Expert Witness

9/2014-Present *Fortinet Inc. vs Sophos Inc., et al*
Client: Fortinet
Counsel: Jordan Jaffe, Kristen Lovin (of Quinn Emanuel)
Issues: Claims construction
Technology: Network security
Status: Declaration submitted; Deposed October 2014; Tech tutorial for Court

3/2014-Present *M2M Solutions vs Motorola Solutions, Telit Communications, and Telit Wireless*
Client: Telit
Counsel: David Loewenstein (of Pearl Cohen)
Issues: Collaborating expert on both patent infringement and invalidity
Technology: Authentication
Status: Reports submitted; Deposition expected in 2015

1/2013-Present *Rmail limited vs. Amazon, Inc. and Paypal*
Client: RMail
Counsel: Lewis Hudnell (of Colvin Hudnell)
Issues: Patent infringement and validity
Technology: Secure email, message authentication
Status: Deposed May 2013; Trial expected in 2015

5/2012-3/2014 *Via Vadis, LLC vs. Skype, Inc.*
Client: Via Vadis, LLC
Counsel: Steven Taylor (of Whiteford, Taylor, Preston)
Issues: Patent infringement
Technology: Peer-to-peer networking
Status: Closed (*No testifying in either deposition or trial*)

Litigation Support

1/2010-Present *Technical (Non-testifying) Expert*
Cases: More than twenty cases involving patents, DMCA, and other IP matters
Technologies: Firewalls, databases, electronic voting, email, wireless protocols, network communications
Services: Analyzing source code, interviewing technical staff, prior art searching of academic and industrial sources, creating claims charts, drafting expert reports, developing infringement and (in)validity theories, rebutting opposing experts, assisting counsel in depositions, preparing demonstrables for trial, training counsel on technical matters

APPENDIX B

Material Considered:

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 <i>Fortinet, Inc. v. Sophos, Inc.</i> , 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 <i>Fortinet, Inc. v. Sophos, Inc.</i> , 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 <i>Fortinet, Inc. v. Sophos, Inc.</i> , 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.