

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

TREND MICRO, INC.
Petitioner,

v.

WEBROOT, INC.
Patent Owner.

IPR2023-00662

U.S. Patent No. 10,599,844

DECLARATION OF SETH JAMES NIELSON

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	QUALIFICATIONS	2
III.	MATERIALS CONSIDERED	12
IV.	LEGAL STANDARDS	14
V.	Technical Background that Would Have Been Known to A POSITA	17
A.	Machine Learning Techniques for Identifying Malware	17
B.	Common Feature: Portable Executable (PE) File Format	20
C.	Support Vector Machines	21
VI.	OVERVIEW OF THE CHALLENGED PATENT	23

A.	THE '844 PATENT SPECIFICATION AND CLAIMS.....	23
B.	PROSECUTION HISTORY.....	27
VII.	LEVEL OF SKILL IN THE ART	28
VIII.	Overview of the Prior art	29
A.	OVERVIEW OF “MAL-ID: AUTOMATIC MALWARE DETECTION USING COMMON SEGMENT ANALYSIS AND META-FEATURES” BY TAHAN ET. AL (“TAHAN”).....	29
B.	OVERVIEW OF “IMPROVING MALWARE DETECTION BY APPLYING MULTI-INDUCER ENSEMBLE” BY MENAHEM ET AL. (“MENAHEM”).....	36
C.	OVERVIEW OF US 2009/0254992 TO SCHULTZ (“SCHULTZ”)	41
IX.	GROUND 1: Claims 1-5, 8, 10-14, 17, and 19 are rendered obvious by TAHAN IN VIEW OF MENAHEM	42
A.	Motivation to Combine Tahan and Menahem	43
B.	Claim 1 is rendered obvious by Tahan in view of Menahem	51
1.	Limitation 1[a]	51
2.	Limitation 1[b]	53
3.	Limitation 1[c]	72
4.	Limitation 1[d]	94
C.	Claim 2 is rendered obvious by Tahan in view of Menahem	101
D.	Claim 3 is rendered obvious by Tahan in view of Menahem	109
E.	Claim 4 is rendered obvious by Tahan in view of Menahem	114
F.	Claim 5 is rendered obvious by Tahan in view of Menahem	120
G.	Claim 8 is rendered obvious by Tahan in view of Menahem	124
H.	Claim 10 is rendered obvious by Tahan in view of Menahem	126
I.	Claims 11-14 are rendered obvious by Tahan in view of Menahem	129
J.	Claim 17 is rendered obvious by Tahan in view of Menahem	131

K.	Claim 19 is rendered obvious by Tahan in view of Menahem	135
X.	GROUND 2: Claims 6-7, 9, 15-16, 18, and 20 are rendered obvious by TAHAN IN VIEW OF MENAHEM in further view of Schultz	136
A.	Motivation to Combine Tahan and Menahem and Schultz	136
B.	Claim 6 is rendered obvious by Tahan in view of Menahem in further view of Schultz	138
C.	Claim 7 is rendered obvious by Tahan in view of Menahem in further view of Schultz	143
D.	Claim 9 is rendered obvious by Tahan in view of Menahem in further view of Schultz	145
E.	Claims 15-16 are rendered obvious by Tahan in view of Menahem in further view of Schultz.....	146
F.	Claims 18, 20 are rendered obvious by Tahan in view of Menahem in further view of Schultz.....	147
XI.	There are no objective indicia of nonobviousness	149
XII.	Conclusion	149

1. I, Seth James Nielson, submit the following declaration (the “Declaration”) in connection with the proceeding identified above.

I. INTRODUCTION

2. I have been retained by counsel for Trend Micro, Inc. (“Trend Micro” or “Petitioner”) as a technical expert in connection with the proceeding identified above. I submit this Declaration in support of Petitioner Trend Micro’s Petition for Inter Partes Review (the “Petition”) of United States Patent No. 10,599,844 (“the ’844 patent”) against Webroot, Inc. (“Webroot” or “Patent Owner”). All “Ex. 10XX” cites herein are to the Exhibits to the Petitions. All “Ex. _” cites followed by a letter are to exhibits to this Declaration. All citations to “Section XX” are internal citations to the sections of this Declaration. All citations to “Paragraph XX” are internal citations to the paragraphs of this Declaration.

3. I have been asked to consider whether the claims of the ’844 patent are obvious in light of the prior art discussed below. The ’844 patent is directed to general application of known machine learning techniques to the identification of malware.

4. In summary, my conclusion is that the claims of the ’844 patent are obvious in light of the prior art discussed below. As set forth in detail in this declaration, I have reached the following conclusions:

- a. Claims 1-5, 8, 10-14, 17, and 19 are rendered obvious by the combination of Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-Features by Tahan, et. al and “Improving Malware Detection by Applying Multi-Inducer Ensemble” by Menahem, et. al.
- b. Claims 6-7, 9, 15-16, 18, and 20 are rendered obvious by the combination of Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-Features by Tahan, et. al, “Improving Malware

Detection by Applying Multi-Inducer Ensemble” by Menahem, et. al, and
U.S. Published Patent Application 2009/0254992 to Schultz, et. al.

II. QUALIFICATIONS

5. I am a subject matter expert in cybersecurity, computer networks, and software engineering. I am the Founder and Chief Scientist of Crimson Vista, a computer security research and engineering company. Furthermore, I hold appointments at the University of Texas at Austin as an Adjunct Associate Professor in the department of Computer Science and as a Cybersecurity Fellow in the Robert Strauss Center for International Security and Law.

6. I have been working with computer networks and associated technology, including parallel processing, since January 2000. My experience includes graduate-level teaching, academic research, industry employment, and consulting practice.

7. A copy of my current C.V. is attached hereto as **Exhibit A**. My C.V. includes all of my relevant education and work experience, a list of all publications that I have authored in the previous ten years, as well as a list of all cases in which, during the previous four years, I have testified as an expert at trial or by deposition. My experience most relevant to the opinions offered in this report is summarized below.

8. I received my B.S. in Computer Science in April of 2000. During my final undergraduate semester, I worked both as a teaching assistant for a Computer Networking course and as a researcher in the Networked Computing Lab. In these capacities, I assisted students in debugging and designing TCP/IP protocol stacks, Address Resolution Protocol implementations, and Remote Procedure Call projects. In the research lab, I collaborated on work related to resource reservations in high performance networking and MPLS analyses. Our investigations into

statistical traffic engineering for bandwidth allocation resulted in a published paper entitled, “Effective Bandwidth for Traffic Engineering.”

9. During the Fall of 2000, I studied for one semester at Brandeis University in the Boston area and, as part of my research, I developed a parallel-processing system for DNA string matching. The system that I created had a central node that managed the search space and solution aggregation. A cluster of available “worker” machines registered with the central node to indicate their availability for parallel processing. The central node would identify subdivisions of the search space that could be farmed out to the worker nodes for processing. I also studied artificial intelligence and machine learning.

10. From 2001 through 2003, I worked as a software engineer at Metrowerks (formerly Lineo, Inc.), where I had substantial responsibilities relating to software architecture, computer networking, and technical project management. In particular, I developed and maintained the GUI for the Embedix SDK (Software Development Kit), created a network communication system between a Windows front-end and Linux back-end, developed an automated system to forward Linux python scripts to a Windows GUI over the network, and developed a network-deployed packaging and automated updating system for client software. To complete these assignments, I wrote tens of thousands of lines of computer code in C++, C, Python, and Perl.

11. As part of my technical training and development at Lineo/Metrowerks, I also gained and employed an increased exposure to computer networking and network security. I tested and evaluated a prototype firewall product, built a custom VPN solution, and trained in the use of web server and mail server administration (including security). I ran my own personal MTA with an IMAP and POP3 server based on these tools under development. I also deployed a Squid web proxy for both caching and content filtering.

12. Another major networking project involved porting the Embedix SDK from Linux to Windows. For this project I used a virtual-machine/networked solution. The underlying SDK engine remained running in a Linux VM while the GUI operated in native Windows. I developed all of the code for this remote communication system.

13. While working at Lineo/Metrowerks, I also returned to BYU to pursue my Master's degree in Computer Science. In addition to the graduate level course work in wireless computer networks and compilers, I pursued graduate research in software engineering topics, with a special emphasis on how programmers think while creating and modifying code. During my course work, I took a special topics class called "Programmer Cognition" as well as a graduate-level neuroscience class from the Psychology department.

14. My research included a study of computer architectural patterns and how those patterns might need to change as programming languages change and evolve. Based on my research, I proposed a concept called "Design Dysphasia," wherein a programmer or software developer becomes trapped in their approach to solving problems based on the paradigms and design approaches of the programming language. My research was published as "Design dysphasia and the pattern maintenance cycle," in the Journal Information and Software Technology August 2006. This work also was a major component of my Master's thesis.

15. Another part of my Master's thesis was the identification of how certain programming language concepts can be "mixed" together. I investigated practical mechanisms whereby the Python programming language could be extended to support features known as "functional programming" and "logic (or declarative) programming." Languages with this mix of features are known as "multi-paradigm" programming languages.

16. After finishing my Master's degree, I moved to Houston TX in 2004 to begin a PhD program at Rice University. I chose to focus my research into topics of network security including the security of peer-to-peer cooperative storage and computation technologies.

17. During the 2004 fall semester of my Ph.D. program at Rice University, I identified a security vulnerability in the Google Desktop Search that could have allowed hackers to compromise users' computers and obtain private information. After contacting Google and assisting them in closing the vulnerability, we published the details of our investigation.

18. In 2005, I completed an internship at Google, where I designed and implemented a solution to privacy loss in Google Web Accelerator. The Google Web Accelerator was designed to increase the speed of browsing the Internet. Once installed on a user's computer, the browser would request all content through a Google Proxy. The proxy performed pre-fetching and extensive caching in order to provide fast and responsive service to the user. At the time of my internship, news reports had identified odd problems in which users of the Accelerator were accessing other individual's private pages. During my internship, I designed and implemented a prototype solution for this issue in C++.

19. In 2005, I published a paper entitled, "A Taxonomy of Rational Attacks." This paper categorized and described the various types of attacks that one might see in a decentralized, peer-to-peer network. When there is no centralized authority, users have to cooperate to obtain service. The term "rational attacks" refers to the economic incentives to not cooperate while still exploiting the system for service.

20. My Ph.D. Thesis was entitled "Designing Incentives for Peer-to-Peer Systems," and it built on this concept. Given a network where participants cannot be forced to cooperate, the operation of said network must induce cooperation by design of the outcomes. In other words, it

must be in each participant's best interest to contribute to the cooperative operation. I conducted experiments including simulated extensions to the BitTorrent peer-to-peer protocol for long-term identities and mechanisms for cooperative anonymity. These systems were predecessors to modern cryptocurrency and distributed ledger technologies.

21. From 2005 through 2008, with the approval of my PhD adviser, I worked as a Security Analyst for Independent Security Evaluators (ISE). Much of my early work was spent developing a software encryption library, including the necessary tests and procedures for FIPS-certification. The encryption library provided advanced operations such as secure data splitting and recovery.

22. In 2009, I went to work full time for ISE as a Security Analyst and later as a Senior Security Analyst. I built a number of advanced projects including a parallel-program for massive code coverage analysis, GPU hardware-accelerated AES encryption, and distributed file-system prototypes.

23. In addition to the software development, I also performed security evaluation services that included port-scanning analyses, security protocol analysis using formal and exploratory methods, and investigated security breaches.

24. I also designed and managed the implementation of a secure communication technology that splits trust between multiple SSL Certificate Authorities (CA), so that if one CA is compromised, the communication stream can still be safely authenticated. My work on the secure communications technology project led to the issuance of multiple patents. In total, I wrote hundreds of thousands of lines of code in C, C++, and Python, including projects where I had to implement the same functionality in two separate languages.

25. In 2011, I began work as a Research Scientist at Harbor Labs and continued with that consulting firm until fall 2015. I worked with a wide range of clients, specializing in network security, network communications, software architecture, and programming languages. I 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 also analyzed security considerations for potential technology acquisitions.

26. Also at Harbor Labs, I reviewed technology and source code for multiple clients related to accusations of theft and/or misappropriation of trade secrets. These engagements included an analysis of C, C++, Java, Python, and other source code languages in high-frequency trading, e-commerce, and other similar systems.

27. I also assessed the security and privacy technologies and policies provided by a third-party vendor to the Center for Copyright Infringement (CCI). CCI represents content owners, such as the RIAA and the MPAA, in finding and reducing piracy online. Because this process necessarily involves collecting information about private individuals by scanning a network for illegal activity, I was asked to investigate and determine that the information collected from online computing devices was adequately safeguarded and protected.

28. For other clients, I have “resurrected” or re-created legacy software systems. For example, I helped one client make remote desktop-sharing/viewing systems from the mid 90’s operational. I helped them identify the most compatible components from an old CVS repository, obtain the necessary legacy hardware and software to rebuild the source code, and diagnose why the separate components weren’t completely compatible with each other. Using tools from the era

(i.e., the mid-90's), I identified and fixed these issues in C++ and Java code, and successfully demonstrated the operational system across a small cluster of networked virtual machines.

29. During my final year at Harbor Labs, I was engaged as the principal consultant with a large biomedical device firm in a twelve-month analysis of the security of their products. Notably, medical devices were for some time not considered significant threats in terms of computer security. However, recent demonstrations by security researchers of the various ways in which a malicious individual might harm a person using a medical device has shifted the thinking in the industry. Accordingly, I was engaged to assist this company in the analysis of their products, their process, and their future roadmap in order to ensure that patients are not harmed. I and my team analyzed design documents, hardware, and a broad range of additional resources in order to expose potential problems. The security of these systems depends, in part, on the architecture and deployment of the networks in which they operate.

30. In December 2015, I left Harbor Labs to assist Ironwood Experts, LLC., as the transitional managing partner. In three months, I helped to establish a new direction, streamline operations, and wrap up difficult negotiations.

31. After handing off management responsibilities at Ironwood, I founded Crimson Vista, Inc. as a boutique computer security engineering company. Similar to the work that I did at Harbor Labs, I continue to provide technical expertise to a wide range of clients in areas of programming languages, computer networks, and network security. My expertise in the area of “security engineering” provides comprehensive analysis, design, and insight into cybersecurity concerns before, during, and after development.

32. For example, I have been retained by a start-up in telecommunications security to provide cryptography expertise and evaluations of their protocols and architectures. My team and

I have prototyped new protocols, written up analyses, and presented to potential partners and investors.

33. Another start-up company retained me for guidance in matters relating to Blockchain and Smart Contracts. This technology is very much dealing with a “fad” phase where there is a lot of misinformation and hype. I guided the start-up company through analyzing where these kinds of technologies would help and where they would not. I have also provided training on Blockchain at the Data Architecture Summit and Enterprise Data World conferences.

34. I have also provided technical guidance to an antitrust team in the United States Department of Justice. Although the technologies and parties are confidential, I can disclose that I provided in-person training on technical topics and analyses of competing security products.

35. More recently, I been retained by clients, including a Fortune 100 financial institution, to provide them with post-data-breach analyses of what went wrong, the impact of the lost data, and guidance on resolution. In these engagements, I provided reverse engineering of the data to demonstrate how an attacker can or would use the compromised information, analyzed software development to determine when the system became vulnerable, and helped identify impacted customers that had been missed in the investigations.

36. I have also been retained as a vCISO (virtual Chief Information Security Officer) for a small company that needs security expertise and guidance in protecting their newly developed intellectual property. I advise the company on security policy, operations, implementations, and training.

37. Through Crimson Vista, I also invest in research and development. Recent projects include engaging with a partner to implement prototypes of communications security protocols for next-generation automobiles. I also gave a talk on “Detecting Malicious Sandboxes” at the

Workshop on Defensive Deception and Trust in Autonomy, in association with the 2018 Naval Applications of Machine Learning Workshop.

38. I was the primary investigator on a government funded research project. The project dealt with automatic recovery from ransomware attacks and was funded by the United States Army. I led a team of student researchers from Johns Hopkins and Brigham Young University in developing my design and implementing a prototype.

39. I am also a contributor to another small-business research project begun by a former student and funded by the National Science Foundation. We are researching the development of a new system for secure data storage that is protected even against “insider” threats, such as a rogue IT administrator. As part of this work, I am managing several interns for Summer 2022 and directing their work on performance, analysis, and vulnerability detection.

40. I also continue to perform a wide range of code reviews for diverse technologies including CAD software, video game systems, digital mobile radios (DMRs), video streaming, and digital rights management (DRM). I am often retained for my expertise in software forensics and have been instrumental in multiple cases to discovering and identifying stolen source code or misappropriated trade secrets.

41. Moreover, I maintain ties to academia. I held adjunct appointments at Johns Hopkins University from 2014 to 2019. From July 2016 to July 2019, I also held an appointment as the Director of Advanced Research Projects in the Johns Hopkins University Information Security Institute.

42. At Johns Hopkins University I taught Network security and Advanced Network Security. I created a custom curriculum and lab experience wherein students developed their own security protocols as a class and then attempted to break their own creations. Students learned how

hard it is to get security right, and how easy it is to find something wrong. I published a paper about the labwork in the Journal of Computer Science Education entitled, “PLAYGROUND: Preparing Students for the Cyber Battleground.”

43. Beyond course instruction, I also mentored Masters students at Johns Hopkins in their capstone projects. These projects included networking security and privacy concerns across a wide range of technologies including cryptography, drone security, iOS security, BitCoin, SSL vulnerabilities, and Twitter “botnets.” My students and I have published multiple papers from these capstone projects.

44. During my tenure as the Director of Advanced Research Projects, I was tasked with developing collaborative research opportunities. Through my efforts, a wide range of student capstones have been executed with partners from the Johns Hopkins Applied Physics Lab or outside corporate partners.

45. For example, we coordinated with the company OnBoard Security to develop better security for anti-collision protocols for air traffic. Students demonstrated the potential issues related to leaving the protocol unsecured and built a working prototype of a secured variant. OnBoard and Johns Hopkins published press releases which were picked up by aviation-focused news sources.

46. I am now an adjunct professor at the University of Texas at Austin. I have taught the undergraduate Network Security and Privacy class in the Computer Science department. I also teach the Introduction to Cybersecurity Technology class in the Law School

47. I am also the co-founder and current director of the Crypto Done Right project. This project is currently hosted by Johns Hopkins University and funded by a grant from Cisco. I am transitioning the project to be hosted by my company Crimson Vista or spun out into a completely

independent non-profit entity. Crypto Done Right is designed to bridge the gap between cryptography SME's and the IT professionals that use it. It provides authoritative guidance on deployment, lifecycle, and management of cryptography in IT systems, software development, and technical management.

48. Finally, I am the author of "Cryptography in Python: Learning Correct Cryptography by Example." I am authoring a second book on computer security that will be published later this year.

III. MATERIALS CONSIDERED

49. I have considered information from various sources in forming my opinions. Besides drawing from approximately two decades of experience in the computer industry, I also have reviewed the '844 patent and its file history, the other documents and references as cited herein, and general technical references. I understand that Counsel for Petitioner have submitted a Petition that reference by exhibit number. For simplicity and convenience, I have adopted the same exhibit numbering.

50. The exhibits are:

- Ex. 1001: U.S. Patent No. 10,599,844 to Schmidtler et al. ('844 patent)
- Ex. 1002: Expert Declaration of Dr. Seth James Nielson
- Ex. 1003: Gil Tahan, Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-Features, Journal of Machine Learning 13, 949-979, (April 13, 2012). ("Tahan")
- Ex. 1004: Eitan Menahem, Improving Malware Detection by Applying Multi-Inducer Ensemble, Computational Statistics & Data Analysis 53, (2009). ("Menahem")

- Ex. 1005: Schultz, et al. US 2009/0254992 “Systems and Methods for Detection of New Malicious Executables”
- Ex 1006: Curriculum Vitae of A.L. Seth Nielson, Ph.D.
- Ex. 1007: Prosecution history of the ’844 patent
- Ex. 1008: J. Zico Kolter, Learning to Detect and Classify Malicious Executables in the Wild, Journal of Machine Learning Research 7, 2721-2744, (December 2006).
- Ex. 1009: Srinivas Mukkamala, Intrusion detection using an ensemble of intelligent paradigms, Journal of Network and Computer Applications 28, 167-182, (2005).
- Ex. 1010: Stuart Russell, Artificial Intelligence: A Modern Approach; The Intelligent Agent Book, 749-751, (Prentice Hall 2) (2003).
- Ex. 1011: Lars Buitinck, API design for machine learning software: Experiences from the scikit-learn project, (September 2013).
- Ex. 1012: Yanfang Ye, SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging. Journal in Computer Virology 5, 283-293, (2009).
- Ex. 1013: Olivier Henchiri, A Feature Selection and Evaluation Scheme for Computer Virus Detection, Sixth International Conference on Data Mining, IEEE, 891–895, (2006).
- Ex. 1014: Jianyong Dai, Efficient Virus Detection Using Dynamic Instruction Sequences, Journal of Computers 4(5), 405–414, (May 2009).
- Ex. 1015: Jeremy Z. Kolter, Learning to Detect Malicious Executables in the Wild, In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (2004).

- Ex. 1016: Thorsten Joachims, Making Large-Scale SVM Learning Practical, LS-8 Report 24, (1999).
- Ex. 1017: Yanfang Ye, Hierarchical associative classifier (hac) for malware detection from the large and imbalanced gray list, Journal of Intelligent Information Systems 35, 1–20, (2010).
- Ex. 1018: Yuval Elovici, Applying Machine Learning Techniques for Detection of Malicious Code in Network Traffic, KI 2007: Advances in Artificial Intelligence, 44–50, (2007).

IV. LEGAL STANDARDS

51. I am not an attorney, and I have relied on instructions from counsel as to the applicable legal standards to use in arriving at my opinions in this Declaration.

52. I have been informed that in a parallel district court litigation the owner of the '844 patent is arguing that the '844 patent claims should be afforded their plain and ordinary meaning. I have also been requested to adopt patent owner's position on claim construction for the purposes of rendering my opinions set forth in this declaration.

53. I understand that a patent may include independent claims and dependent claims. An independent claim stands by itself and includes the limitations it recites. A dependent claim can depend on (a) an independent claim, or (2) another dependent claim. I understand that a dependent claims include all of the limitations it recites plus the limitations recited in the claim from which it depends.

54. I have been informed and understand that the subject matter of a patent claim is obvious if the differences between the subject matter of the claim and the prior art are such that the subject matter of the claim as a whole would have been obvious at the effective filing date to

a person having ordinary skill in the art to which the subject matter pertains. I have also been informed that the framework for determining obviousness involves considering the following factors: (i) the scope and content of the prior art; (ii) the differences between the prior art and the claimed subject matter; (iii) the level of ordinary skill in the art; and (iv) any objective evidence of non-obviousness.

55. I further understand that the claimed subject matter would have been obvious to one of ordinary skill in the art if, for example, it results from the combination of known elements according to known methods to yield predictable results, the simple substitution of one known element for another to obtain predictable results, the use of a known technique to improve similar devices in the same way, the application of a known technique to a known device ready for improvement to yield predictable results, or choosing from a finite number of identified, predictable solutions with a reasonable expectation of success. I have also been informed that the analysis of obviousness may include recourse to logic, judgment, and common sense available to the person of ordinary skill in the art that does not necessarily require explication in any reference. I understand that so-called “secondary considerations of non-obviousness,” must be considered in an obviousness analysis. I understand that an analysis cannot use hindsight in determining whether a patent claim is obvious. I understand that secondary considerations of non-obviousness include: (a) a long-felt but unresolved need for the invention; (b) commercial success of the invention; (c) copying of the invention; (d) praise and recognition of the invention by others; (e) licensing of the rights to the invention; and (f) unexpected results.

56. I understand that obviousness must be determined with respect to the challenged claim as a whole.

57. I understand that a challenged claim can be invalid for obviousness over a combination of prior art references if a reason existed (at the effective filing date of the Patent in question) that would have prompted a person of ordinary skill in the art to combine prior art elements in the way required by the claim in question. I understand that this requirement is sometimes called “motivation to combine.” I further understand that the prior art references themselves might provide a motivation to combine, but that sometimes common sense as would have been known to a POSITA can provide a reason to link two or more prior art references. I also understand that market demand, rather than scientific literature, often drives innovation, so that market forces can give rise to a motivation to combine. I also understand that known work in a field of endeavor might prompt variations of it to use in the same field or a different one based on design incentives or market forces if the variations would have been predictable to a POSITA. I also understand that a need or problem known in the field of endeavor can provide a reason for combining elements in the manner claimed.

58. I understand that in order for a claim to be found invalid based upon a modification or combination of prior art references, a POSITA must have had a reasonable expectation of success in modifying or combining the prior art.

59. I further understand that claims are invalid as obvious if they merely recite combinations of, or obvious variations of, prior art teachings employing the understanding and knowledge of a POSITA and motivated by the general problem that the inventor faced at the time. The prior art references themselves, or any need or problem known in the field of endeavor at the time of the invention can provide a motivation to combine the elements from prior art references or to attempt obvious variations on them in the claimed manner.

60. I understand that something is “inherent in” the prior art—and thus taught by it—it if necessarily follows from the prior art’s explicit disclosures.

61. I understand that the filing date of a U.S. patent is generally the filing date of the U.S. application that issued as a patent. I also understand, though, that U.S. patent claims may be granted an earlier effective filing date if the patent owner properly claimed priority to an earlier patent application that fully supports the claims.

V. TECHNICAL BACKGROUND THAT WOULD HAVE BEEN KNOWN TO A POSITA

62. In this section of my Declaration, I discuss some of the background and context for the alleged invention in the ’844 Patent that would have been known to a POSITA as of the filing date of the ’844 Patent, which I understand to be May 12, 2015. Any discussion of the background technologies in this section are written to be as understood by a POSITA of that date.

A. Machine Learning Techniques for Identifying Malware

63. Malware, which is a harmful or malicious computer program, is a significant problem for computer users. Defensive organizations, such as anti-malware companies, invest significant resources into identifying and blocking malware. One of the more common approaches is to use “signatures” that contain patterns for identifying specific, known malware. The problem, of course, is that these systems cannot block new malware that has not been seen before and for which no identifying signature exists.

Detection of known malware is commonly performed by using anti-virus tools. These tools detect the known malware using signature detection methods (White, 1999; Dickinson, 2005). When a new malware is released, the anti-virus vendors need to catch an instance of the new malware, analyze it, create a new signature and update their clients. During the period between the appearance of a new unknown malware and the update of the signature-base of the anti-virus clients, millions of computers are vulnerable to the new malware.

(Ex. 1004 at p. 1483).

64. Machine learning is one tool used since at least the late 1990's to try and detect previously unseen malware. One approach is to build, using machine learning techniques, a “classifier” that is designed to analyze a new (previously unseen) sample and determine if it is malicious. This technology was widely studied and used by the time the '844 Patent was filed. (See Ex. 1003, p. 951). Although there are many ways of building such a classifier, many of the approaches followed a three-part approach.

[I]n recent years many researchers have been using machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files.

The techniques use three distinct stages:

1. Feature Extraction for file representation—The result of the feature extraction phase is a vector containing the features extracted...
2. Feature Selection (or feature reduction)—During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded...
3. The last phase is creating a classifier using the reduced features vector created in phase 2 and a classification technique. Among the many classification techniques... [is] support vector machines (SVM)

(Ex. 1003, pp. 950-951).

65. For many techniques, the third phase is a “training”/“classification” phase. Typically, a machine learning technique, such as Support Vector Machines (SVM), takes as input a set of known malware and a set of known benign software and creates/trains a classifier based on the training set (e.g., which is used to classify a file as malicious or benign (as discussed below)).

66. Sometimes, the technique used to create (or train) a classifier is called an “inducer.” There are many types of inducers that can be used to create a classifier and the type chosen can have an impact on the effectiveness of the classifier. “By its nature, the measured performance

(e.g., accuracy) of a machine learning classifier is influenced by two main factors: (1) the inducer algorithm that is used to generate the classifier (e.g., Artificial Neural Network, Decision Tree, Naïve Bayes); and (2) the types of features that are used to represent the instances.” (Ex. 1004 at p. 1484).

67. Once the classifier is created, it generally takes an unknown sample as input and spits out a classification. However, classifiers can be retrained based on subsequent inputs.

A new classification rule set is generated at the model generator 240 by running the data mining algorithm on the new data set that contains the borderline cases along with their correct classification (as determined by expert analysis), and the existing training data set. (Ex. 1005 at [0110]).

68. One important point is that **features** (e.g., that can represent a set of features (e.g., a specific set of data identifying features within a given object) or a feature vector) are extracted for both **training** the classifier, and for **classifying** the unknown samples. The general process works like this:

- a. First, the process ingests a sample set of files that is already labeled (e.g., already known and marked as “malicious” or “benign”)
- b. Second, the process extracts a set of features from each of the training files
- c. Third, a machine learning algorithm uses the features of the sample files to calculate/create (i.e., “train”) a classifier
- d. Fourth, the process extracts a set of features from a new file (that does not yet have a label as “malicious” or “benign”) and inputs this set of features into the (trained) classifier
- e. Fifth, the classifier outputs a label (e.g., “malicious” or “benign”) for the previously unseen feature vector

- f. Sixth (and optionally), new files with known labels (e.g., “malicious or “benign”) are used by the machine learning algorithm to further train (i.e., “retrain”) the classifier.

69. It should be noted that the retraining may use files that have been classified by the classifier. However, it is almost always wrong to retrain the classifier using the classifier’s prediction as the correct label. In other words, if the classifier says that a new file is “malicious”, this is not enough to then retrain the classifier using this file-label pair. Rather, a human or other system (perhaps another AI system) gives an independent label, which may be the same as or different than the original classification.

70. It is also worth mentioning that because a classifier is constructed from a specific machine learning algorithm, it is not unusual to identify it as that “type” of classifier. So, for example, a classifier trained by Support Vector Machines (SVM) techniques is often called an SVM Classifier. (See, e.g., Menahem generally).

B. Common Feature: Portable Executable (PE) File Format

71. In machine learning, one common source for features used when training a classifier is file metadata. Because executable files for the Windows file system are a common target for malware, metadata of these types of files are also common sources of classifier features. Because the use of PE file format data is discussed by multiple prior art references and the ’844 Patent (see, e.g., ’844 Patent at Table 1.1), I include the following brief summary.

72. Portable Executable (PE) “is a file format for executables, object code, dynamic link library files (DLLs), and font library files, among other examples.” (Ex. 1001, 3:41-43). The format has been used on Windows since at least 1999. (See Ex. 1005, [0044]). PE files have a header (data at the beginning of the file) that contains a wide range of metadata.

73. For example, PE file headers can include informational metadata:

- a. Machine type
- b. Date created
- c. Information about tools that built the file

74. PE file headers can also include information used by the computer in executing the file such as:

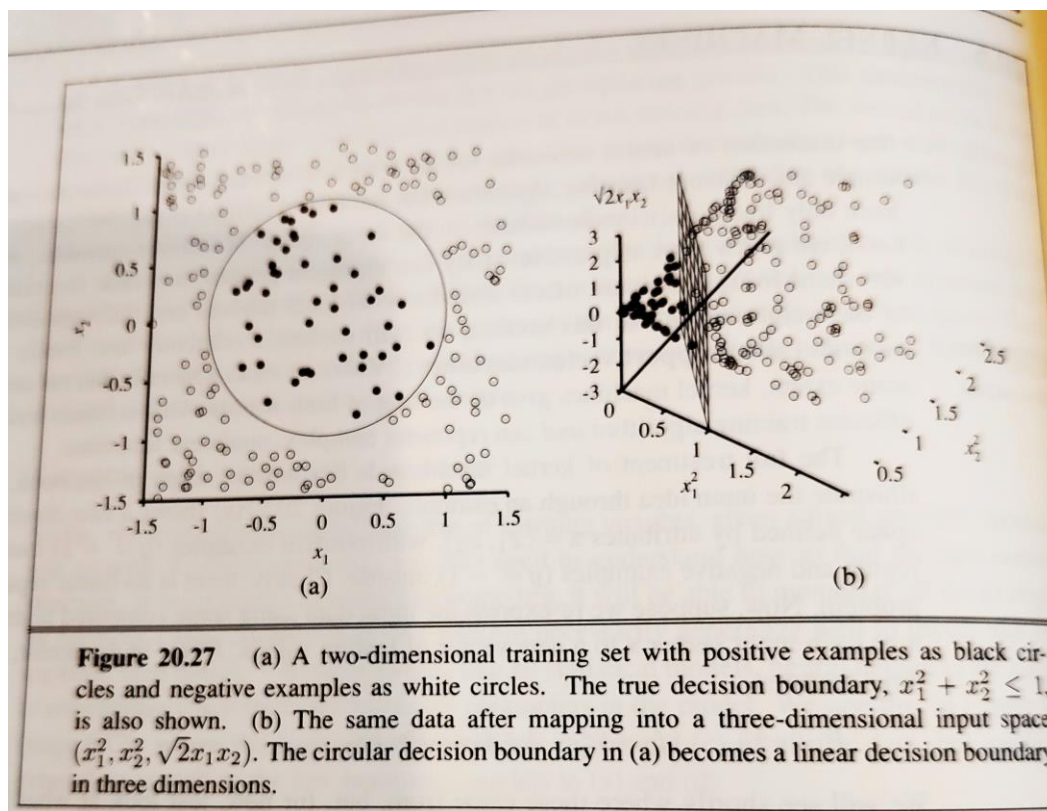
- a. File characteristics
- b. Memory address offset to the start of code
- c. Sizes of various parts of the file
- d. The number of sections the file is divided into

75. Some of the metadata is encoded as a “flag” (e.g., indicating on/off or yes/no values (e.g., Boolean values)). A flag is a value that can, in some way or another, indicate the presence of a characteristic. In other words, if the flag value is present, the characteristic is present in the file; otherwise it is not. For example, the file characteristics are encoded as flags and include flags for things like “Can this file be run?” For example, this flag is denoted by Windows as “IMAGE_FILE_EXECUTABLE_IMAGE”. If the value associated with this flag is present in the characteristics part of the file, the Windows operating system will know that the file can be executed. Other flags (e.g., on/off values, which could be considered Boolean values) are found through the PE file header for various purposes including section types, loading parameters, and debug information.

C. Support Vector Machines

76. One machine learning technique for creating/training a classifier is called Support Vector Machines or SVM. Support Vector Machines, despite the name “Machine” being in the name, are machine learning algorithms. The basic idea is to create a binary classifier (it classifies an input into one category or another) that separates data linearly. Imagine a graph with points

plotted at X,Y coordinates and where every point is labeled as either category 1 or category 2. Each point represents a feature vector and the label is the known classification (e.g., a feature vector for known malware). For some types of data, the points for each category can be separated by a line running through the graph. However, there are data sets in which there is no linear separator—for example, if the data from one category was more or less in a circle on the graph and all the data for the second category surrounded it. This is shown in Figure 20.27(a) from the textbook “Artificial Intelligence, a Modern Approach, Second Edition” (copyright 2003, attached as Ex. 1010), reproduced hereinbelow:



77. As can be seen in this figure, because category one is inside a circle, and category two surrounds it, there is no way to draw a line through the data to separate it.

78. However, the data can be converted into **three-dimensional** space in such a way as to have a linear separator in the form of a plane that divides the data. This is illustrated in Figure

20.27(b) from the same book, as shown above. Note that the 3-D coordinates were chosen in order to create this projection from the 2-D coordinates. “[I]f data are mapped into a space of sufficiently high dimension, then they will always be linearly separable.” (Ex. 1010, p. 2).

79. The goal of the SVM algorithm is to find an “optimal linear separator – the one that has the largest margin between it and the positive examples on one side and the negative examples on the other.” (*Id.* at p. 2). But it turns out that working in the higher-dimensions can be computationally costly. SVM solves these problems with some mathematical tricks (using a concept called “kernel functions”) that enable the learning (i.e., the training of the classifier) in the higher dimensions at a lower computational cost. (*Id.* at pp. 2-4).

80. I note that this is a simplified overview and should not be considered a thorough or detailed explanation of SVM.

VI. OVERVIEW OF THE CHALLENGED PATENT

A. THE '844 PATENT SPECIFICATION AND CLAIMS.

81. U.S. Patent No. 10,599,844 (“the '844 Patent”) relates to detection of malware in executable files. ('844 Patent at Abstract). The '844 Patent emphasizes that it analyzes previously unknown executables. (*Id.* at 1:7-10). The alleged invention of the '844 Patent allegedly identifies threats within an executable file even if the file is encrypted or compressed.

The present disclosure is able to detect whether an executable file is harmful or benign before the executable file is actually executed on a processing device. In examples, machine learning processing applies a classifier to evaluate an executable file based on static points data collected from the executable file. The classifier is trained from a collection of data comprising known malicious files, potentially unwanted files and benign files. The classifier is designed and trained such that it can handle encrypted and/or compressed files without decrypting and/or decompressing the files. ('844 Patent at 2:7-14).

82. Within the specification, the '844 discusses approaches for identifying malware that were well known in the research community. For example, the specification describes

operations that follow the three-step approach identified as common to many machine-learning approaches , as I explain in Sections V.A. and VIII.A. That is, the '844 patent discloses collecting/extracting features, narrowing the features, and then training a classifier using a machine learning method.

- a. Collecting/extracting features for a feature vector ('844 Patent at 8:22-48).
- b. Narrowing features used ('844 Patent at 12:54-63).
- c. Training a classifier ('844 Patent at 9:41-67).

83. The classifiers disclosed in the '844 include SVM-trained classifiers and combination methods wherein multiple classifiers are combined into an ensemble classifier. ('844 Patent at 9:41-67). This approach is called an ensemble and is also well known and is described in many sources including Tahan and Menahem, as described below. (See, e.g., my explanation in Sections VIII.A-B. and IX.B). The '844 Patent specification discusses subdividing inputs into different types of encodings that can be used with component SVM classifiers that are fed into the ensemble classifier. ('844 Patent at 9:41-67).

84. The '844 Patent discusses generating feature vectors from static data points. ('844 Patent, 5:19-6:22, 8:35-9:40). The Patent discusses, as an example, PE header data and n-grams as different examples of static data points. (See, e.g., '844 Patent Table 1.1, 5:58-6:22). The claims of the '844 Patent require that the static data points represent “predefined character strings.” (See, e.g., '844 Patent Claim 1).

85. As discussed in my Technical Background Section V., the machine learning techniques are applied to create/train a classifier. The classifier itself takes in a feature vector from an input and outputs a label (e.g., classification or determination as to whether a file is harmful or malware). However, the '844 Patent sometimes refers to the entire system that extracts the data,

generates feature vectors from static data points, trains the classifier, and performs the classification as the “classifier” and sometimes as the “classifier component.”

86. For example, the '844 specification states, “a learning classifier (*e.g.*, *learning classifier component 104*) may be trained to identify features in the extracted data to identify malicious files.” ('844 Patent at 6:64-66, emphasis added). The classifier component is broader than just the algorithms for classification and include the classifier training components. As stated in the specification, “The learning classifier component 104 utilizes programming operations or machine-learning processing to train/re-train the learning classifier to uniquely evaluate any file/file type.” ('844 Patent at 8:11-14). In fact, the specification states that, “a learning classifier [is] built by the learning classifier component 102.” ('844 Patent at 7:32-33). The specification also explains that the learning classifier component uses the extracted features to train and build the learning classifier. “As an example, the learning classifier 104 applies... machine learning processing to evaluate static data points extracted from a file to be analyzed. In doing so, the learning classifier component 104 builds a learning classifier based on... the extracted static data points of a file and the information used to train/re-train the learning classifier.” ('844 Patent at 7:46=53).

87. The '844 Patent specification sometimes uses the term “classifier” to mean the broader “classifier component” (as well as classifier algorithm, which may be part of the broader classifier component). For example, in describing feature vector generation, the specification explains: “In one example, operation 306 comprises generating at least one feature vector from the plurality of static data points extracted **using** the learning classifier... In order to generate the feature vector **for** the learning classifier, data may be parsed and encoded for machine learning processing.... In one example, generation of the feature vector may comprise selectively setting

features of the learning classifier based on the one or more of static data points extracted.” (’844 Patent at 12:48-53, emphasis added). A POSITA would have recognized that the term “classifier” (e.g., in the specification) sometimes means the broader “classifier component.” For example, in the quote above, a POSITA would have understood that the “learning classifier” in the phrase “generating at least one feature vector from the plurality of static data points extracted using the learning classifier” would be a broader classifier component, which performs certain tasks such as extracting static data points and “generat[ing] the feature vector *for the learning classifier*,” wherein this latter instance of “learning classifier” refers to a classifier algorithm. (See *id.*, emphasis added).

88. For example, the *Scikit-Learn* project was an open source machine learning library for Python, which I have used myself. (See Ex. 1011). This library provides machine learning classifiers that can be trained and used for classification within the Python language. The classifier software in this library is organized into modules that perform operations using APIs (Application Programming Interfaces). Through APIs, a programmer can configure the classifier, perform feature selection, train the model, and so forth. Within Scikit-Learn, all of these APIs are grouped together into a single module for each type of classifier. Specifically, this grouped classifier module includes all of the elements required to perform classification including feature extraction, training, and so forth. As explained in the original white paper entitled, “API design for machine learning software: experiences from the scikit-learn project” (published September 2013):

The estimator interface is at the core of the library. It defines instantiation mechanisms of objects and exposes a “fit” method for learning a model from training data. ***All supervised and unsupervised learning algorithms (e.g., for classification, regression or clustering) are offered as objects implementing this interface. Machine learning tasks like feature extraction, feature selection or dimensionality reduction are also provided as estimators.***

(Ex. 1011 at p. 5, emphasis added).

89. A POSITA would have understood that the above-discussed grouped (classifier) module in Scikit-Learn is substantially similar in concept to the above-discussed broader “classifier component” (e.g., “learning classifier component”) in the ’844 Patent. (See *id.*).

B. PROSECUTION HISTORY

90. The application that issued as the ’844 Patent, 14/709,875, was filed on May 12, 2015. (See Ex. 1007, p. 2). I understand that during Prosecution of the 14/709,875 application, there were several non-final rejections, various amendments to the claims, and multiple Examiner interviews before the patent issued on March 24, 2020. (See Ex. 1007, p. 1).

91. I also understand that in the final amendment filed September 26, 2019, Applicants amended the independent claims to add the subject matter recited in claim 5. Specifically, the independent claims were amended to recite “... generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range....” (See Ex. 1007, p. 85). In the Notice of Allowance, the Examiner provided an examiner’s statement of reasons for allowance that restated this claim element including the amendment language. (See Ex. 1007, pp. 18-19).

92. During Prosecution, Patent Owner also explained that the static data points described in the claims represented predefined character strings. Patent Owner argued that the alleged invention needed to be able to compare data from a sample file to predefined character

strings. “Notably, as discussed in the examiner interview, the **fixed-length file chunks do not represent predefined character strings. Instead, each fixed-length file chunk is, itself, a random (e.g. undefined) character string having no initial significance to the system** of Wojnowicz. The entropy analysis performed on the random character string does not include a comparison of the random character string to a predefined character string. In contrast, claim 1 as amended specifies that the static data points represent one or more predefined character strings. The system of the present application is configured to identify the predefined character strings and the values associated therewith. **Based on the presence of the predefined character strings and/or the corresponding values, the system determines whether a file is malicious. The portion of the file that does not correspond do [sic] the predefined character strings is not analyzed by the system.** Thus, the system of the present application provides at least processing/computational improvements over the system of Wojnowicz.” (Ex. 1007, pp. 155-156).

93. Patent Owner further reinforced this requirement and stated that “[t]he portion of the file that does not correspond do [sic] the predefined character strings is not analyzed by the system.” (*Id.*).

VII. LEVEL OF SKILL IN THE ART

94. I understand that Petitioners submit that a person of ordinary skill in the art (“POSITA”) at the time of the earliest claimed priority date for the ’844 patent would have had a bachelor’s degree in computer science, or a related field, or an equivalent technical degree or equivalent work experience, and an additional one to two years of education or experience in the field of computer security topics such as malware/exploit detection and/or machine learning. More education can supplement practical experience and vice versa.

95. I am at least a person of ordinary skill in the art and I have worked with persons of ordinary skill in the art as part of my professional responsibilities.

VIII. OVERVIEW OF THE PRIOR ART

A. OVERVIEW OF “MAL-ID: AUTOMATIC MALWARE DETECTION USING COMMON SEGMENT ANALYSIS AND META-FEATURES” BY TAHAN ET. AL (“TAHAN”)

96. The published journal article, “Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-Features,” by Gil Tahan, Lior Rokach, and Yuval Shahar (“Tahan”) was published in the Journal of Machine Learning Research in April 2012. The abstract of the paper states, “This paper proposes several novel methods, based on machine learning, to detect malware in executable files without any need for preprocessing, such as unpacking or disassembling.” (Ex. 1003, Abstract).

97. Tahan provides a summary of at least a portion of the State of the Art as of the time of its publication. For example, Tahan discloses:

[I]n recent years many researchers have been using machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files.

The techniques use three distinct stages:

1. Feature Extraction for file representation—The result of the feature extraction phase is a vector containing the features extracted...
2. Feature Selection (or feature reduction)—During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded...
3. The last phase is creating a classifier using the reduced features vector created in phase 2 and a classification technique. Among the many classification techniques... [is] support vector machines (SVM).

(Ex. 1003, pp. 950-951).

98. With respect to the first phase (e.g., “Feature Extraction for file representation”), Tahan discusses many approaches to feature extraction. One of such approaches relates to

extraction of features such as PE headers or PE file format fields. (See also my explanation regarding PE file format fields in Section V.B.)

Portable Executable File Format Fields—information embedded in Win32 and Win64-bit executables. The information is necessary for the Windows OS loader and application itself. Features extracted from PE executables may include all or part of the following pieces of information: attribute certificate—similar to checksum but more difficult to forge; date/time stamp; file pointer—a position within the file as stored on disk; linker information; CPU type; Portable Executable (PE) logical structure (including section alignment, code size, debug flags); characteristics—flags that indicate attributes of the image file; DLL import section—list of DLLs and functions the executable uses; export section—which functions can be imported by other applications; resource directory—indexed by a multiple-level binary-sorted tree structure (resources may include all kinds of information. For example, strings for dialogs, images, dialog structures; version information, build information, original filename, etc.); relocation table; and many other features.
(Ex. 1003, p. 950)

99. In the second of these three phases (e.g., “Feature Selection (or feature reduction)”), Tahan describes the feature-reduction process as the standard way that machine learning is applied to classify malware.

In addition, in recent years many *researchers have been using machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files...*

Feature Selection (or feature reduction)—During this phase the vector created in phase 1 is evaluated and **redundant and irrelevant features are discarded**. Feature selection has many benefits including: improving the performance of learning modules by reducing the number of computations and as a result the learning speed; enhancing generalization capability; improving the interpretability of a model, etc. Feature selection can be done using a wrapper approach or a correlation-based filter approach (Mitchell, 1997). Typically, the filter approach is faster than the wrapper approach and is used when many features exist. **The filter approach uses a measure to quantify the correlation of each feature, or a combination of features, to a class.** The overall expected contribution to the classification is calculated and selection is done according to the highest value. **The feature selection measure can be calculated using many techniques, such as gain ratio (GR); information-gain (IG); Fisher score ranking technique (Golub et**

al., 1999) and hierarchical feature selection (Henchiri and Japkowicz, 2006)."

(Ex. 1003, pp. 950-951, emphasis added).

100. In the third phase a classifier that has been trained on a set of prepared feature vectors is then able to classify a new vector not previously seen. Tahan discloses the use of support vector machines. "In addition, in recent years many researchers have been using machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files... The last phase is creating a classifier using the reduced features vector created in phase 2 and a classification technique. Among the many classification techniques, most of which have been implemented in the Weka platform (Witten and Frank, 2005), the following have been used in the context of benign/malware files classification... support vector machines (SVMs)..." (Ex. 1003, pp. 950-951).

101. The overview Tahan provides is a good summary of the art and existing technologies at the time Tahan was published. Moreover, it is the same framework in which Tahan discloses and on which Tahan builds another machine learning classification technique called "Mal-ID." As described in the following paragraphs, Mal-ID involves feature selection, feature extraction, feature reduction, training of a classifier, and evaluation of an executable file using a machine learning classifier.

102. Tahan provides the following overview of Mal-ID: "The basic method (Mal-ID) is a new static (form-based) analysis methodology that uses common segment analysis in order to detect malware files. By using common segment analysis, Mal-ID is able to discard malware parts that originate from benign code. In addition, Mal-ID uses a new kind of feature, termed meta-feature, to better capture the properties of the analyzed segments." (Ex. 1003 at p. 949). As stated,

Tahan identifies improvements to the art in terms of features and feature selection: meta-features and common segment analysis.

103. Within the Tahan reference, the term “segment” is sometimes used instead of the term “n-gram.” A POSITA would have recognized that these terms have the same meaning. For example, Tahan defines N-grams as “segments of consecutive bytes from different locations within the executables of length n.” (Ex. 1003 at p. 951). Later, Tahan explains that segments are created by dividing “file F into S segments of length L” (Ex. 1003 at p. 958). A POSITA would have known that a “segment of length L” is a “segment of consecutive bytes from [a location] within the executable...”

104. Moreover, Tahan also equates n-grams and segments in discussing a desired length of the subsequences. Tahan explains that “[i]n many cases 3- to 8-byte n-gram cannot represent even one line of code composed with a high level language. In fact, we showed in a previous paper (Tahan et al., 2010) that an n-gram should be at least 64 bytes long to uniquely identify a malware...” (Ex. 1003 at p. 952, emphasis added). In the next paragraph, Tahan switches to using the term “segment” while discussing the same concept: “[n]ote that, when a segment represents at **least one line of code developed using a high level language**; it can address the second deficiency of using short features that may be meaningless when considered alone.” (*Id.*, emphasis added).

105. Thus, a POSITA would have recognized that Tahan’s n-grams and segments have the same meaning, and that one of Tahan’s improvements is the use of longer n-grams¹ to better capture sequences of data, such as lines of code.

¹ Tahan also uses shorter 3-gram features to identify the segments. “Each segment comprises a number of 3-grams... Three features can be derived for each segment: Spread, MFG, and Entropy... spread approximates the maximum number of occurrences of a segment within different relative locations... the maximum total number of file-groups [(MFG)] that contain the (footnote continued)

106. Tahan also discloses analyzing segments in order to determine which segments are common to benign files and malicious files. The concept is related to features in that features that show up in both malware and benign files are preemptively removed, reducing the feature set. Tahan explains that “[a]ll applications... that were developed using a higher level development platforms [sic]... contain common code and resources that originate from common code and resource libraries. Since most malware are also made of the same common building blocks, we believe it would be reasonable to discard the parts of a malware that are common to all kinds of software, leaving only the parts that are unique to the malware” (Ex. 1003 at p. 952).

107. Once the common segments are removed, Tahan identifies segments that are common to the malware (based on a training set). “In other words, when given an unclassified file, the first step would be to detect the file segments that originated from the development platform or from a benign third party library (termed here the Common Function) and then disregard those segments. Finally, the remaining segments would be compared to determine their degree of resemblance to a collection of known malwares. If the resemblance measure satisfies a predetermined threshold or rule then the file can be classified as malware.” (Ex. 1003 at p. 955). Tahan collects the “common” (benign) segments into a Common Function Library (CFL) and segments that only appear in malware into a Threat Function Library (TFL). (*Id.*).

108. Sequences/segments in the CFL and TFL are then pre-defined binary sequences/segments used by the classifier and against which other files are compared. Tahan explains that segments should be discarded, for example, if “it may appear in benign files.” (Ex.

segment.” (Ex. 1003 at p. 957). Tahan explains that even though the 3-grams are short sequences, using “meta-features” such as MFG and spread, “Mal-ID can refer to relatively long sequences (64 bytes)...” (Ex. 1003 at pp. 957-958).

1003 at p. 958). On the other hand, “segments that actually reside in the malware are left in the segment collection.” (Ex. 1003 at p. 959).

109. Tahan indicates that the approach works for encrypted and unencrypted files. “[W]e hypothesize that the proposed Mal-ID method performs well in a mixed environment where both obfuscated (including compressed or encrypted) and plain executable files exist... we have independently reconfirmed the validity of the above observation using our method...” (Ex. 1003, p. 975).

110. The following figure from Tahan illustrates combining these various elements together. During the setup phase, the system identifies a common function library (CFL) that identifies sequences/segments that are in benign software and the threats function library (TFL) that identifies sequences/segments that are only in malware (based on training data). During the detection phase, the potential malware is broken into segments and certain features are considered, including entropy. Segments are further categorized by whether they are in the CFL or TFL.

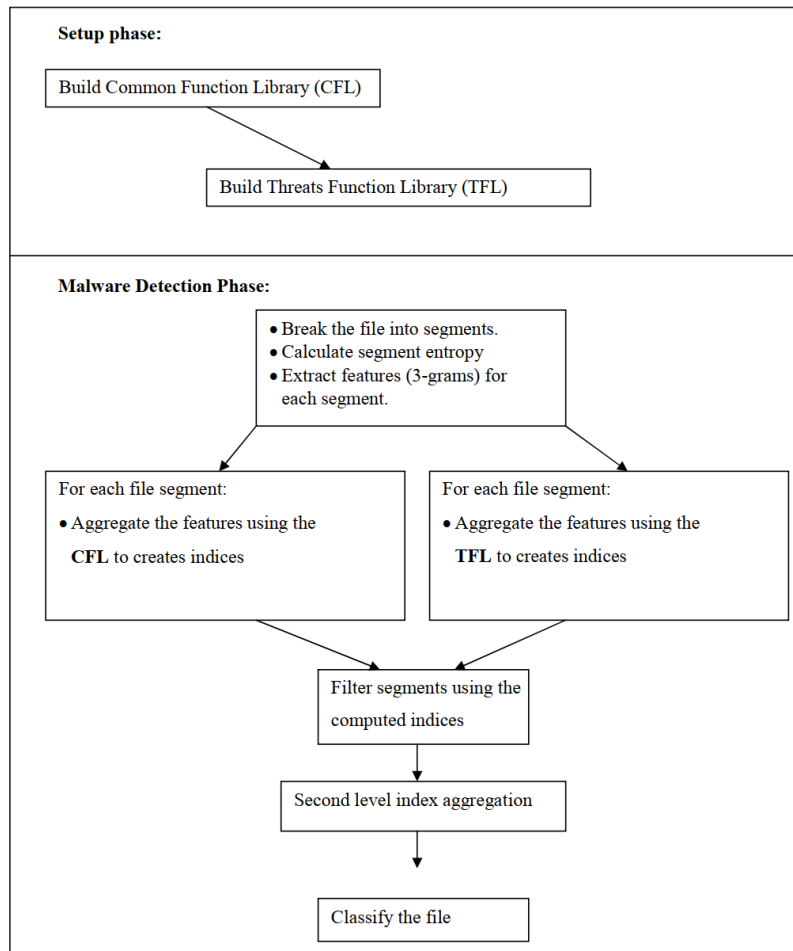


Figure 1: The Mal-ID method for detecting new malware applications.

111. One of the reasons Tahan considers entropy of a segment is to decide whether to ignore it. As Tahan explains, “the entropy measure of the bytes within a specific segment candidate... the entropy feature is also used to enable to identification of compressed areas (such as embedded JPEG images) and long repeating sequences that contain relatively little information.” (Ex. 1003 at p. 957). Moreover, “The entropy thresholds are set to disregard compressed areas (such as embedded JPEG images) and long repeating sequences that contain relatively little information.” (Ex. 1003 at p. 958). A POSITA would have known that “compressed” and “encrypted” data have nearly identical entropy scores and that this filtration of high-entropy segments also removed encrypted segments as well.

112. In addition to using Mal-ID alone to classify files, Tahan also uses Mal-ID in combination with other classifiers. Tahan explains that “[w]e attempted to improve the Mal-ID basic method by using Mal-ID features with various classifiers...” (Ex. 1003 at p. 959). Specifically, Tahan combines Mal-ID with three other classification technologies known as “Rotation Forest,” “C4.5”, and a “multilayer perception.” (Ex. 1003 at p. 961). In evaluating these combinations, Tahan noted that no one approach was superior in all cases. “For each [classifier] method we have pinpointed its strong points and suggested cases where it should be preferred over the others.” (Ex. 1003 at p. 976).

113. Although Tahan does not expressly disclose combining Mal-ID with SVM classifiers, Tahan reviews a wide range of machine learning research that applies SVM classifiers to identify malware. For example, Tahan identifies “Ye et al. (2009)”, “Dai et al. (2009)”, “Kolter and Maloof (2004)”, “Kolter and Maloof (2006)”, and “Henchiri and Japkowicz (2006)”, all of which are examples of research into SVM classifiers and malware detection. (Ex. 1003 at pp. 951-953). This further demonstrates that using SVM classifiers for detecting malware was well-known in the art and to a POSITA before the filing date of the ’844 Patent. (See *id.*)

B. OVERVIEW OF “IMPROVING MALWARE DETECTION BY APPLYING MULTI-INDUCER ENSEMBLE” BY MENAHEM ET AL. (“MENAHEM”)

114. The published journal article, “Improving Malware Detection by Applying Multi-Inducer Ensemble” by Eitan Menahem, Asaf Shabtai, Lior Rokach, and Yuval Elovici (“Menahem”) was published in the Computational Statistics and Data Analysis journal 19 October 2008. Menahem explains that different types of classifiers are used to detect malware, including combinations of classifiers called “ensembles” (which Menahem also refers to as a super classifier). Menahem explains, “In order to benefit from multiple different classifiers, and exploit their strengths we suggest using an ensemble method that will combine the results of the individual

classifiers into one final result to achieve overall higher detection accuracy... The main goal is to find the best combining method...” (Ex. 1004, Abstract).

115. In addition to combining different types of classifiers, Menahem also uses feature vectors comprised of different types of features, each encoded in an appropriate way. “We collected a large set of 22,735 benign files and 7690 malware that were identified as Win32 executable and DLLs. For each file **three types of features** were extracted: n-grams, Portable Executable (PE) features and Function-based features as described bellow. [sic]” (Ex. 1004, p. 1487, emphasis added).

116. Binary n-grams in the Menahem article are extracted directly from the binary representation of a file using a “sliding window” algorithm for each of the different values of n (3, 4, 5, and 6). These features are encoded based on the frequency of the n-gram. (Ex. 1004, p. 1487).

117. With reference to the Portable Executable (PE) data, Menahem teaches extracting and encoding data about the presence of data (a Boolean value), counts of data, and other references to contents/structure:

We statically extracted different Portable Executable (PE) format features that represented information contained within each Win32 PE binary (EXE or DLL)... The information extracted by this tool is as follows (total of 88 attributes):

- Information extracted from the PE-header that describes the details about physical structure of a PE binary (e.g., creation\modification time, machine type and file size)
- Optional PE-header information describing the logical structure of a PE binary (e.g., linker version, section alignment, code size and debug flags)
- Import Section: which DLLs were imported and which functions from the imported DLLs that were used
- Export Section: which functions were exported (if the file being examined is a DLL)
- Resource Directory: resources used by a given file (e.g., dialogs and cursors)

- Version Information: (e.g., internal and external name of a file, version number)
(Ex. 1004, pp. 1487-1488).

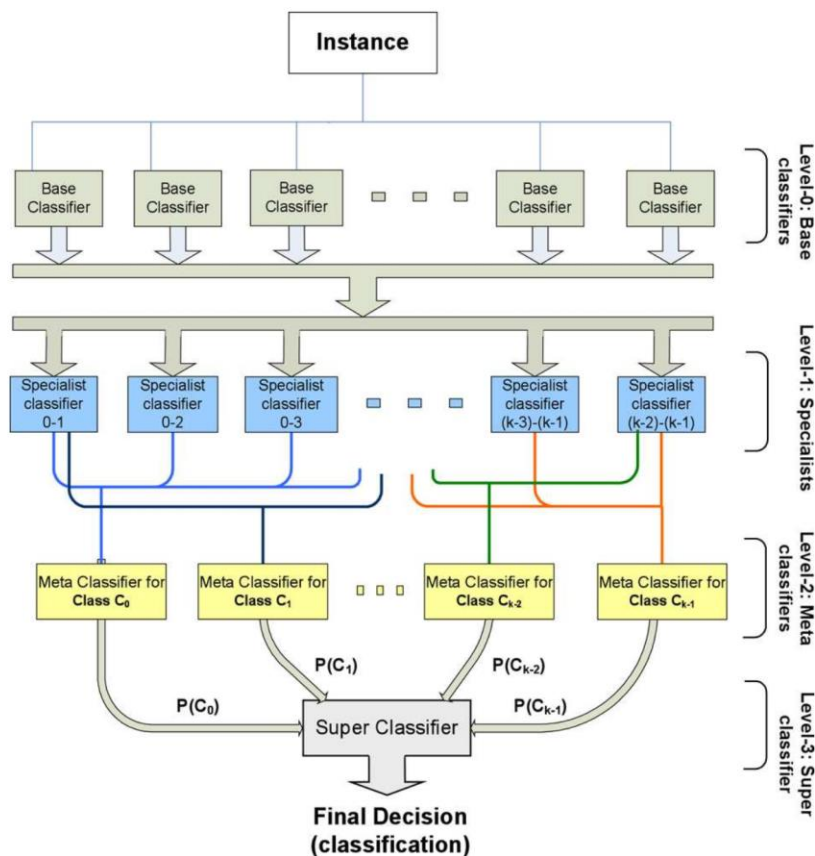
118. With respect to “function-based features,” Menahem explains that “we use a j48 decision tree classifier in order to mark the beginning and ending of functions residing in the binary representation of the files.” (Ex. 1004 at p. 1488). Using these function marks, Menahem extracts the following attributes/features, which are derived from an executable file:

- Size of file
 - File’s entropy value
 - Total number of detected functions
 - Size of the longest function that was detected
 - Size of the shortest function that was detected
 - Average size of detected functions
 - Standard deviation of the size of detected functions
 - Number of functions divided into fuzzy groups by its length in bytes (16, 24, 40, 64, 90, 128 and 256 bytes)
 - Function ratio — the proportion of number of detected functions from the overall size of the file
 - Code ratio — the proportion of the size (in bytes) of detected functions from the overall size of the file.
- (Ex. 1004, p. 1488).

119. Menahem discloses and uses a mix of different classifiers to create a composite “ensemble” classifier. Ensemble classification uses multiple component/base classifiers, potentially built using different techniques, or potentially with different data. When classifying an input, the data is processed in some way by the component/base classifiers and a final decision, or answer, is rendered by the aggregate classifier. For example, one approach is voting, wherein each component classifier gets a vote and the answer with the highest number of votes is selected as the answer for the overall classifier. “Ensemble of classifiers [sic] was evaluated on intrusion detection tasks as well. Mukkamalaa et al. (2005) used the Majority Voting schema to ensemble the results of multiple classifiers trained to detected intrusions within network traffic. The performance of the ensemble results was compared to the performance of the individual classifiers:

SVM, MARS and three types of ANNs (RP, SCG and OSS)...” (Ex. 1004 at p. 1484). In this example from Menahem, the ensemble of combined classifiers (such as SVM and other classifiers) were used and performance of such an ensemble classifier was compared with that of each of the individual classifiers. Menahem reports, “[i]t is shown that the simple majority voting schema [of the ensemble classifier] improves the accuracy of the detection.” (Ex. 1004 at p. 1485).

120. Menahem’s experiments set forth different approaches for creating the ensemble classifier (which Menahem also refers to as a super classifier). As Menahem summarizes, “we have examined seven different combination methods... The goal was to identify the best combination method.” (Ex. 1004 at p. 1493). One of the seven methods Menahem discloses is called “Troika.” A visual representation of how component/base classifiers are used in Troika to make a determination is shown in the figure reproduced hereinbelow:



(Ex. 1004, p. 1486).

121. In the figure above, the “base classifiers” are individual classifiers. Menahem discloses different types of classifiers (e.g., “inducers” that are used to create/train classifiers) that are used as base classifiers: “[i]n the experiment we had used five different inducers as our base-classifiers: C4.5... KNN... VFI... OneR... and Naïve Bayes... Each inducer belongs to different family [sic] of classifiers.” (Ex. 1004 at p. 1487). As Menahem explains, these classifiers are representative but by no means “the best” or “only” classifiers that should be used in an ensemble. As Menahem further explains, “[f]uture work may include the involvement of more inducers for base-classifiers (we had only 5, while there are plenty more to choose from)...” (Ex. 1004 at p. 1494).

122. Menahem also refers to the use of Support Vector Machines (SVMs), including in ensemble classifiers. “Ensemble of classifiers [sic] was evaluated on intrusion detection tasks as well. Mukkamalaa et al. (2005) used the Majority Voting schema to ensemble the results of multiple classifiers trained to detected intrusions within network traffic. The performance of the ensemble results was compared to the performance of the individual classifiers: SVM, MARS and three types of ANNs (RP, SCG and OSS)...” (Ex. 1004 at p. 1484).

123. Menahem further discloses that classifiers could be trained on these different feature sets and then combined into an ensemble classifier. “In addition, in the hereby described experiment, we evaluated different inducers on the same datasets. In future experiments we would like to evaluate the performance of inducers that are trained on different features.” (Ex. 1004, p. 1494). Menahem further explains that ensemble classifiers benefitted from diversity. “When generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently... The outcome of this

observation is that one should use multiple different classifiers... combining all the results, using an intelligent combination method into one final result.” (Ex. 1004 at p. 1484).

124. Furthermore, Menahem concludes that a properly chosen ensemble classifier performs better than individual base classifiers. Menahem’s results indicate that the “Troika” method showed the most improvement in classification accuracy, compared to other ensemble methods (e.g., “Stacking”) and compared to “the best classifier chosen using cross-validation” (i.e., the most accurate individual base classifier):

From the results we learn that ***in terms of accuracy, and AUC, Troika is probably the best choice***; it is better than Stacking, the only other ensemble in the group of meta-classifier ensembles. ***It is also better than the best classifier chosen using cross-validation.*** The benefit of using Troika is double; first, its AUC and accuracy are very high. Second, when using a multi-class dataset, its AUC improves in about 8%, producing a model with lower FPR and a nice capability of identifying the malware’s specific class. (Ex. 1004, p.1494, emphasis added).

C. OVERVIEW OF US 2009/0254992 TO SCHULTZ (“SCHULTZ”)

125. U.S. Published Patent Application 2009/0254992 to Schultz et al. (“Schultz”) describes “[a] system and methods for detecting malicious executable attachments at an email processing application of a computer system using data mining techniques.” (Schultz at Abstract). The executables are classified into classes such as malicious and benign, but Schultz also discloses that classification can depend on a probability exceeding a threshold. (*Id.*).

126. The “data mining techniques” of Schultz include various ML (machine learning) classifiers. The disclosed classifiers work with some of the same features disclosed and used by Tahan and Menahem, such as strings taken from PE headers. (Ex. 1005, [0045]-[0055], [0068]).

127. With respect to the classifying based on a threshold, Schultz discloses that the threshold used to block malware can be chosen based on pre-determined false positive rates. Schultz explains, and it was known in the art, that a “Receiver Operating Characteristic” (ROC) is

a score derived from both false-positives and false-negatives. The ROC can be graphed illustrating the trade-off between false-positives (or false-negatives) and detection rate because improving one generally makes the other worse. Schultz described picking the threshold value based on these curves:

FIG. 11 displays the plot 400 of the detection rate 402 vs. false positive rate 404 using Receiver Operation Characteristic curves, as described in K. H. Zou et al., “Smooth Non-Parametric ROC Curves for Continuous Diagnostic Tests. Statistics in Medicine, 1997. Receiver Operating Characteristic (ROC) curves are a way of visualizing the **trade-offs between detection and false positive rates**. In this instance, the ROC curve shows how the data mining method (illustrated in dashed line 405) can be configured for different environments. **For a false positive rate less than or equal to 1% the detection rate would be greater than 70%, and for a false positive rate greater than 8% the detection rate would be greater than 99%**. Thus, more secure settings would select a threshold setting associated with a point on the data mining line towards the right (indicated by arrow 408), and applications needing fewer false alarms should choose a point towards the left (indicated by arrow 406). (Ex. 1005, [0121], emphasis added).

128. Moreover, Schultz discloses updating a classifier when accuracy of its classification of an executable file (e.g., as malicious or not) was a borderline case but was subsequently confirmed by a human expert.

A new classification rule set is generated at the model generator 240 by running the data mining algorithm on the new data set that contains the borderline cases along with their correct classification (as determined by expert analysis), and the existing training data set. (Ex. 1005 at [0110]).

IX. GROUND 1: CLAIMS 1-5, 8, 10-14, 17, AND 19 ARE RENDERED OBVIOUS BY TAHAN IN VIEW OF MENAHEM

129. As I describe in greater detail hereinbelow, a POSITA would have found Claims 1-5, 8, 10-14, 17, and 19 rendered obvious by Tahan in view of Menahem.

A. Motivation to Combine Tahan and Menahem

130. A POSITA would have been motivated to combine the teachings of Tahan and Menahem. As an initial, threshold matter, both references are explicitly directed toward solving the same problem: classifying previously unknown malware using machine learning techniques. For example, Tahan “proposes several novel methods, based on machine learning, to detect malware in executable files.” (Ex. 1003 at p. 949). Similarly, Menahem explains that “[t]he main goal is to find the best [machine learning] combining method for the task of detecting malicious files...” (Ex. 1004 at p. 1483).

131. As explained in my background section on the reference, Tahan’s disclosure defines, describes, and evaluates a new classification mechanism called “Mal-ID,” which (as I discussed above for Section VIII.A) is built upon techniques known in the art. “The basic method (Mal-ID) is a new static (form-based) analysis methodology that uses common segment analysis in order to detect malware files. By using common segment analysis, Mal-ID is able to discard malware parts that originate from benign code. In addition, Mal-ID uses a new kind of feature, termed meta-feature, to better capture the properties of the analyzed segments.” (Ex. 1003 at p. 949).

132. As also explained above, Tahan discloses combining Mal-ID with other classification methods (including those that were known in the art): “[w]e attempted to improve the Mal-ID basic method by using Mal-ID features with various classifiers...” (Ex. 1003 at p. 959). Specifically, Tahan combines Mal-ID with three other classification technologies known as “Rotation Forest,” “C4.5”, and a “multilayer perception.” (Ex. 1003 at p. 961). Tahan further identifies that other classifiers were known in the art to be used for classifying malware, including Support Vector Machines (SVM). (See Ex. 1003 at pp. 951-953: “the following have been used in the context of benign/malware files classification: ... support vector machines (SVMs)

(Joachims, 1999)”, “Ye et al. (2009) [taught] ... a classification model is constructed from SVM ensemble with bagging”, “Dai et al. (2009) [found] ... In the evaluation the SVM model performed slightly better than the C4.5 model”; see also my analysis in Section VIII.A.).

133.

134. A POSITA would have further recognized (e.g., based on Tahan’s disclosures) that different combinations of different machine-learning approaches resulted in no one universal “best” combination. Tahan summarizes thus: “For each method we have pinpointed its strong points and suggested cases where it should be preferred over the others.” (Ex. 1003 at 976). A POSITA would have been motivated to look for and try different techniques, classifier combinations, and feature combinations, in order to identify other solutions or methods for additional “strong points and suggested cases where [such methods] should be preferred over others” (and to achieve efficient and effective machine learning processes for detecting harmful executable files overall) (Ex. 1003 at p. 976).

135. As I discussed above, the overview Tahan provides is a good summary of the techniques known in the art and existing technologies at the time Tahan was published. (See Sections V.A., VIII.A.). Moreover, it is the same framework in which Tahan discloses and on which Tahan builds another machine learning classification technique called “Mal-ID.” (See *id.*) As described in Sections V.A., VIII.A., and for Claim 1, Mal-ID involves feature extraction, feature selection/reduction, training of a classifier, and evaluation of an executable file using a machine learning classifier (e.g., concepts and approaches also known and used in the art). Moreover, as discussed above, Tahan discloses combining Mal-ID with other classifiers and techniques known in the art. Furthermore, Tahan’s techniques such as Mal-ID and those known in the art were directed toward solving the same problem: classifying/detecting previously

unknown malware using machine learning techniques. (See *id.*; see generally Ex. 1003). Thus, a POSITA would have understood that Tahan's techniques are built upon (and are compatible with) techniques known in the art, and thus would have been motivated to combine techniques known in the art (e.g., including those discussed in Tahan and Menahem) with Tahan's techniques (including Mal-ID) to build an effective method/system for detecting malware using machine learning processes and techniques. (See *id.*)

136. A POSITA would have looked to a reference like Menahem, which also combines machine-learning algorithms, in order to obtain these advantages. Menahem discloses an "ensemble" combination of machine learning approaches/classifiers: "[i]n order to benefit from multiple different classifiers, and exploit their strengths we suggest using an ensemble method that will combine the results of the individual classifiers into one final result to achieve overall higher detection accuracy..." (Ex. 1004, Abstract).

137. Menahem discloses combining multiple classifiers with multiple types of features. Like Tahan, Menahem works with n-gram sequences, but Menahem also uses data from executable file headers (PE headers). This data is either raw strings or data derived from the strings (such as the number of functions, which is based off counting function definition strings). Menahem describes examples of PE data collected:

We statically extracted different Portable Executable (PE) format features that represented information contained within each Win32 PE binary (EXE or DLL)... The information extracted by this tool is as follows (total of 88 attributes):

- Information extracted from the PE-header that describes the details about physical structure of a PE binary (e.g., creation\modification time, machine type and file size)
- Optional PE-header information describing the logical structure of a PE binary (e.g., linker version, section alignment, code size and debug flags)
- Import Section: which DLLs were imported and which functions from the imported DLLs that were used

- Export Section: which functions were exported (if the file being examined is a DLL)
 - Resource Directory: resources used by a given file (e.g., dialogs and cursors)
 - Version Information: (e.g., internal and external name of a file, version number)
- (Ex. 1004, pp. 1487-1488).

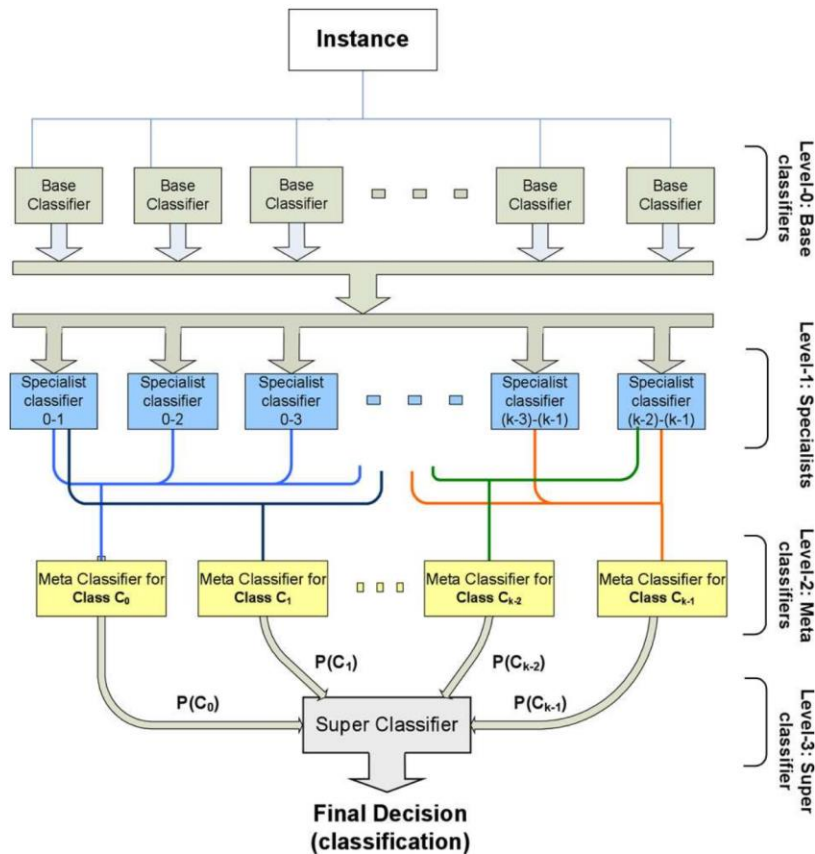
138. Menahem teaches using an additional type of features called “function-based features.” Menahem explains that “we use a j48 decision tree classifier in order to mark the beginning and ending of functions residing in the binary representation of the files.” (Ex. 1004 at p. 1488). Using these function marks, Menahem extracts the following attributes/features, which are derived from an executable file:

- Size of file
 - File’s entropy value
 - Total number of detected functions
 - Size of the longest function that was detected
 - Size of the shortest function that was detected
 - Average size of detected functions
 - Standard deviation of the size of detected functions
 - Number of functions divided into fuzzy groups by its length in bytes (16, 24, 40, 64, 90, 128 and 256 bytes)
 - Function ratio — the proportion of number of detected functions from the overall size of the file
 - Code ratio — the proportion of the size (in bytes) of detected functions from the overall size of the file.
- (Ex. 1004, p. 1488).

139. Menahem classifies the different types of data into different encodings in feature vectors, each of which is processed by a base classifier and the results are aggregated for final classification. Menahem also discloses using these features to train different kinds of base classifiers. Specifically, Menahem discloses five different types of classifiers (inducers) that are used as base classifiers: “[i]n the experiment we had used five different inducers as our base-classifiers: C4.5... KNN... VFI... OneR... and Naïve Bayes... Each inducer belongs to different family [sic] of classifiers.” (Ex. 1004 at p. 1487). Menahem further discloses that its ensemble

classifier technique builds on prior research about ensemble classifiers that used component/base classifiers (inducers) known in the art to be useful for malware detection, including Support Vector Machines (SVM). (See Ex. 1004 at p. 1484: “the datasets [of Kolter and Maloof (2006)] were evaluated using several inducers: IBK, Naïve Bayes, Support Vector Machines (SVM), Decision Trees, Boosted Naïve Bayes, Boosted SVMs, and Boosted Decision Tree algorithms”, “Mukkamalaa et al. (2005) used the Majority Voting schema to ensemble the results of multiple classifiers trained to detect intrusions within network traffic. The performance of the ensemble results was compared to the performance of the individual classifiers: SVM, MARS and three types of ANNs (RP, SCG and OSS)”); see also my analysis in Section VIII.B.).

140. Menahem teaches taking these base classifiers and combining them to create the ensemble classifier. As Menahem summarizes, “we have examined seven different combination methods... The goal was to identify the best combination method.” (Ex. 1004 at p. 1493). One of the seven methods Menahem discloses is called “Troika.” A visual representation of how component/base classifiers are used in Troika to make a determination is shown in the figure reproduced hereinbelow:



(Ex. 1004, p. 1486).

141. A POSITA would have been motivated to combine Tahan’s Mal-ID classifier (and other classifiers such as those known in the art) with other classifiers into an ensemble as taught by Menahem. As Menahem explains, “[w]hen generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently... The outcome of this observation is that one should use multiple different classifiers... combining all the results, using an intelligent combination method into one final result.” (Ex. 1004, p. 1484). Furthermore, as I explained in Section VIII.B., a POSITA would understand from Menahem that properly creating such an ensemble classifier will yield better classification accuracy than using even the most-accurate component/base classifier. (See Ex.

1004, p. 1494: “It [the best ensemble classifier] is also better than the best [base] classifier chosen using cross-validation”; see also my explanation in Section VIII.B.).

142. Tahan already teaches (and provides motivation for) combining Mal-ID with other classifiers (e.g., those known in the art) and found that the results were better in certain circumstances. Tahan explains that “[w]e attempted to improve the Mal-ID basic method by using Mal-ID features with various classifiers...” (Ex. 1003 at p. 959). Specifically, Tahan combines Mal-ID with three other classification technologies known as “Rotation Forest,” “C4.5”, and a “multilayer perception.” (Ex. 1003 at p. 961). In evaluating these combinations, Tahan noted that no one approach was superior in all cases. “For each [classifier] method we have pinpointed its strong points and suggested cases where it should be preferred over the others.” (Ex. 1003, p. 976).

143. A POSITA would have been motivated to build on Tahan’s combination of Mal-ID with other classifiers (e.g., those known in the art) using Menahem’s more advanced ensemble combinations. Tahan describes the strengths of a particular classifier, and Menahem teaches how to combine multiple base classifiers (including, a POSITA would understand, Tahan’s teachings) into better-performing ensemble classifiers.

144. A POSITA would have further found it obvious to use Support Vector Machines (SVM) as a component/base classifier, given its success in the art of machine learning techniques for detecting malware (as cited by both Tahan and Menahem) and its compatibility with Menahem’s ensemble classifier technique (which technique builds on prior research specifically using SVM as a component/base classifier of an ensemble classifier).

145. A POSITA also would have been motivated to take advantage of Menahem’s teachings on effective feature selection/reduction. Tahan teaches the importance of features, as I

explained in my overview of the reference (see Section VIII.A.). As I explained therein and for Claim 1, Tahan discusses selecting longer sequences of n-grams, as well as ignoring “common” data found in both benign and malicious code and also discusses feature selection/reduction techniques known in the art. (See my analysis for Section VIII.A. and for limitation 1[b]; see also Ex. 1003 at p. 951). A POSITA would have recognized the importance of building on these teachings of Tahan. A POSITA would have looked to a reference like Menahem that also teaches that it was well known in the art that the effectiveness of a classifier was determined in part by a feature vector and its features (and thus the importance of an effective feature selection/reduction process): “By its nature, the measured performance (e.g., accuracy) of a machine learning classifier is influenced by... the types of features that are used to represent the instances.” (Ex. 1004 at p. 1484).

146. Moreover, with respect to an ensemble classifier, Menahem discloses that the “combined” features for a super classifier/ensemble classifier could be composed of a combination of different features, each for its own classifier. “In future experiments we would like to evaluate the performance of inducers that are trained on different features.” (Ex. 1004, p. 1494). Menahem identified that this was desirable. “When generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently” and contribute to improved performance of a super classifier that is an ensemble of classifiers. (Ex. 1004, p. 1484; see *id.*, p. 1494; see also my analyses and explanations in Sections V.A., VIII.B., and for Claim 1). Thus, a POSITA would have been motivated to combine the teachings of Tahan and Menahem regarding, *inter alia*, methods and approaches regarding classifiers and features (e.g., to use/try different classifiers (e.g., in an ensemble) along with

different types of features taught by the two references to achieve an improved machine learning classification process to detect harmful executable files).

147. A POSITA would have recognized that by using Menahem's ensemble, including mixed feature vectors, multiple machine learning techniques disclosed by Tahan could be used together to have a more reliable result across all cases without having to determine which one should be used for a given case. As Menahem states, "In order to benefit from multiple different classifiers, and exploit their strengths we suggest using an ensemble method that will combine the results of the individual classifiers into one final result to achieve overall higher detection accuracy." (Ex. 1004 at p. 1483).

148. Moreover, a POSITA would have understood that the methods of Tahan and Menahem are compatible, because they teach similar approaches, methods, and concepts, such as extracting features from an executable file, training a classifier, and using a classifier to determine if an executable file is harmful (e.g., malware). (See my analyses for Sections V.A-B., VIII.A-B., and Claim 1 in IX.B.).

149. Therefore, a POSITA would have been motivated to combine the teachings of Tahan and Menahem.

B. Claim 1 is rendered obvious by Tahan in view of Menahem

150. Each of the limitations of Claim 1 of the '844 patent is rendered obvious by Tahan in view of Menahem as described below.

1. Limitation 1[a]

151. Limitation 1[a] of the '844 Patent reads:

A computer-implemented method comprising:

152. I understand that the preamble of a patent claim is not always considered limiting. However, to the extent that the preamble of Claim 1 of the '844 Patent is considered limiting, Tahan discloses the preamble of Claim 1.

153. Tahan discloses a computer-based system/method for identifying malware, a portion of which is illustrated in the following figure from Tahan. During the setup phase, the computer system identifies a common function library that identifies sequences that are in both malware and benign software and the threats function library that identifies sequences that are only in malware (based on training data). During the detection phase, the potential malware is broken into segments and features are calculated for each segment, including entropy and other computer-based classifications. (See Ex. 1003, p. 956).

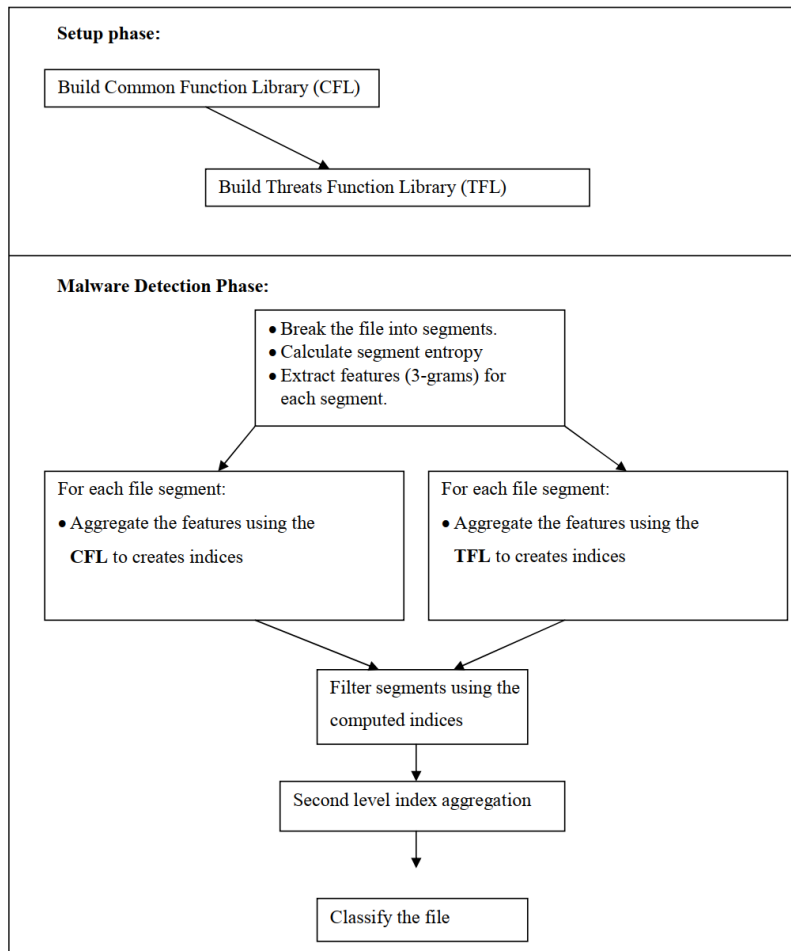


Figure 1: The Mal-ID method for detecting new malware applications.

154. Moreover, Tahan explains, “This paper proposes several novel *methods*, based on machine learning, to detect malware in executable files without any need for preprocessing, such as unpacking or disassembling.” (Tahan at Abstract, emphasis added).

155. Thus, I conclude that Tahan discloses the preamble of Claim 1.

2. Limitation 1[b]

156. Limitation 1[b] of the ’844 Patent reads:

extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file;

157. Tahan alone or in view of Menahem discloses limitation 1[b] of the '844 Patent. I have decomposed this limitation into three requirements.

“extracting a plurality of static data points from an executable file”

158. This is the first requirement. The '844 Patent describes “static data points” as “a point of reference used to evaluate an executable file. As examples, static data points include, but are not limited to: header information, section information, import and export information, certificate information, resource information, string and flag information legal information, comments, and/or program information (e.g., APIs and DLLs), among other examples.” ('844 Patent, 5:62-6:1).

159. The '844 Patent also describes “types” of static data points, such as “numeric values,” “nominal values,” “strings/byte sequences,” and “Boolean values”. Table 1.1, reproduced hereinbelow, provides a few examples of what the '844 Patent considers to be examples of these types of static data points.

TABLE 1.1

Numeric values	Nominal values	Strings/Byte sequences	Boolean values
File size	initialize	Comments	Address Of Entry Point Anomaly
linker version	Un-initialize	company name	Image Base Anomaly
code size	entry point	file description	Section Alignment Anomaly
OS version	subsystem	internal name	Size Of Code Mismatch Anomaly
image version	file subtype	legal copyright	Low Import Count Anomaly
subsystem	language	original file	Entry Point Anomaly
version			
file version	file flags masks	private build	certificate Validity
number			
product version	file flags	product name	Certificate Exception
number			
size of heap	file OS	special build	Code Characteristics Anomaly
size of stack	file type	product version	Code Name Anomaly
size of image	machine type	file version	Count Anomaly
PE header time	PE type	package code	Data Characteristics Anomaly
Section Entropy	section counts	product code	Data Name Anomaly
Sections count	DLL count	export DLL name	Export Exception
	DLL functions	assembly version	Large Number of DLLs Anomaly
	data directory	Certificate Issuer	flag DLL Name Anomaly
	export count	Certificate	Number of Functions
		Subject	Anomaly
	Earliest Data Byte	Imports	Function Name Anomaly
	resources	Exports	PE Header Anomaly
	resources language	Section Names	High Section Count Anomaly
	resource Encoding	Non-resource	PE Magic Validity
		section strings	
	resource code		Resource Exception
	page		
	resource size		VR Code Ratio Anomaly
	DLL		Import Exception
	characteristics		

(’844 Patent, Table 1.1).

160. Tahan discloses extracting features (from executable files) that include all four types of “static data points” identified by the ’844 Patent specification: numeric values, nominal values, character/binary strings, and Boolean values. As I explained with respect to the background topic on machine learning techniques, features are extracted for both training the classifier and for classifying unknown samples. (See Section V.A.).

161. For example, Tahan explains, “machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files . . . use three distinct stages:

1. **Feature Extraction** for file representation—The result of the *feature extraction phase* is a

vector containing the features extracted. An **executable content** [e.g., executable file] is reduced or transformed into a more manageable form such as: **Strings . . . Portable Executable File Format Fields . . . n-gram . . . Opcode n-gram . . .**” (Ex. 1003, pp. 950-951, emphasis added). Tahan also explains, “The Mal-ID basic is a **feature extraction process** followed by a simple static decision rule. It operates by analyzing **short segments extracted from the file examined**. Each **segment comprises a number of 3-grams** depending on the length of the segment (e.g., a segment of length 4 bytes is comprised from two 3-grams that overlap by two bytes). Three features can be derived for each **segment: Spread**, MFG, and **Entropy**.” (*Id.*, p. 957, emphasis added). I discuss these extracted features (“static data points”) below.

162. A major component of Tahan’s disclosures are binary strings extracted for analysis. Tahan discloses extracting these strings, known as n-grams: “n-gram—segments of consecutive bytes from different locations within the executables of length n. Each n-gram extracted is considered a feature” (Ex. 1003, p. 951). Furthermore, “In this paper, the term segment refers to a sequence of bytes of certain size that was extracted from an executable file.” (Ex. 1003, p. 950). A POSITA would have understood that while Tahan uses several terms to refer to binary strings (e.g., “n-gram”, “segment”, “sequence of bytes”), Tahan uses all of them to teach extracting features (“static data points”) from executable files. For example, Tahan teaches that the n-grams, which are binary strings, can also represent character strings, such as specific lines of code. Tahan explains its motivation for using “segments” (e.g., as extracted features in machine learning analysis to detect malwares):

In many cases 3- to 8-byte n-gram cannot represent even one line of code composed with a high level language. In fact, we showed in a previous paper (Tahan et al., 2010) that an n-gram should be at least 64 bytes long to uniquely identify a malware. As a result, current techniques using short n-gram rely on complex conditions and involve many features for detecting malware files.

The goal of this paper is to develop and evaluate a novel methodology and supporting algorithms for detecting malware files by using common segment analysis. In the proposed methodology, we initially detect and nullify, by zero patching, benign segments and therefore resolve the deficiency of analyzing files with segments that may not contribute or even hinder classification. Note that, when **a segment** represents at **least one line of code developed using a high level language**; it can address the second deficiency of using short features that may be meaningless when considered alone.

(Ex. 1003, p. 952, emphasis added).

163. Thus, Tahan teaches that, even though the extracted features (or “static data points”) are just n-gram sequences from within the file, the sequences represent data, such as lines of code and other strings (also referred to as “segments”). (See also Section VIII.A. regarding extraction of features (“static data points”) including n-grams, segments, and sequence of bytes, and the discussion on Tahan equating these terms.)

164. In another example, Tahan discloses that an alternative form of n-grams (i.e., binary strings) was used in the art. Tahan describes these as “Opcode n-gram[s]” wherein an “Opcode is a CPU specific operational code that performs specific machine instruction. Opcode n-gram refers to the concatenation of Opcodes into segments.” (Ex. 1003, p. 951). A POSITA would have understood that Opcode n-grams are a binary character sequence, and that they represent the same concept as what the ’844 patent refers to as “static data points include[ing] . . . program information [such as] APIs...” (’844 Patent at 5:63-6:1).

165. With respect to “static data points” that are numeric values, Tahan discloses extracting (from executable files) similar metrics related to file size that are used to identify “relative locations” in a file. “3-gram-relative-position-within-file: 224 entries, each of 64 bits. A bit value of 1 in a cell (i, j) indicates the appearance of 3-gram i in the jth internal segment of a file (assuming the file is divided into 64 equal length segments).” (Ex. 1003, p. 957). A POSITA would have understood that Tahan’s teaching about locating a segment within a file (see, e.g., “224

entries, each of 64 bits”, *Id.*) teaches the same concept as the numeric value “File size” (static data point) disclosed in Table 1.1 of the ’844 Patent specification. Furthermore, these are used in deriving an additional value about the file called the “spread”. As Tahan explains, “[e]ach segment comprises a number of 3-grams... spread approximates the maximum number of occurrences of a segment within different relative locations....” (Ex. 1003, p. 957).

166. Tahan also discloses another numeric value: entropy. Tahan defines entropy as “the entropy measure of the bytes within a specific segment candidate... the entropy feature is also used to enable to identification of compressed areas (such as embedded JPEG images) and long repeating sequences that contain relatively little information.” (Ex. 1003, p. 957). A POSITA would have understood this “measure” of the bytes to be a value associated with character sequences. A POSITA would have further understood this to teach the same concept as taught by the ’844 Patent specification’s disclosure of “Section entropy”: a numeric “static data point[] identified for analysis of an executable file” (’844 Patent at 20-21 and Table 1.1).

167. Tahan further discloses other examples of “static data points” of various types that are extracted from executable files. Specifically, Tahan discloses that it was very common in the art to extract and use data from executable file headers, such as Portable Executable (PE) file headers, as features, which are “static data points”:

Portable Executable File Format Fields—information embedded in Win32 and Win64-bit executables. The information is necessary for the Windows OS loader and application itself. Features extracted from PE executables may include all or part of the following pieces of information: attribute certificate—similar to checksum but more difficult to forge; date/time stamp; file pointer—a position within the file as stored on disk; linker information; CPU type; Portable Executable (PE) logical structure (including section alignment, code size, debug flags); characteristics—flags that indicate attributes of the image file; DLL import section—list of DLLs and functions the executable uses; export section—which functions can be imported by other applications; resource directory—indexed by a multiple-level binary-sorted tree structure (resources may include all kinds of

information. For example, strings for dialogs, images, dialog structures; version information, build information, original filename, etc.); relocation table; and many other features.
(Ex. 1003, p. 950)

168. A POSITA would have recognized that Tahan describes many “[f]eatures extracted from PE executables” that the ’844 Patent specification also teaches. For example, the following table illustrates some parallels a POSITA would have recognized:

Tahan (Ex. 1003, p. 950)	’844 Patent specification at 5:62-6:1	’844 Patent specification at Table 1.1
attribute certificate	certificate information	
date/time stamp		• PE header time
linker information		• linker version
CPU type		• machine type
Portable Executable (PE) logical structure (including section alignment, code size, debug flags)		• code size
DLL import section—list of DLLs and functions the executable uses	import and export information program information (e.g., APIs and DLLs)	• DLL count • DLL functions • DLL characteristics
export section—which functions can be imported by other applications	import and export information	• export DLL name
resource directory—indexed by a multiple-level binary-sorted tree structure (resources may include all kinds of information. For example, strings for dialogs, images, dialog structures; version information, build information, original filename, etc.)	string and flag information resource information	• resources • product version • private build • special build • internal name

169. Tahan indicates that the use of PE data was common in the art. In particular, Tahan identified research from “Schultz et al. (2001)”, “Elovici et al. (2007)”, “Ye et al. (2008)”, “Ye et al. (2009)”, and “Ye et al. (2010)” as all using PE data in some form or another as features. (Ex. 1003, p. 953).

170. A POSITA would have recognized that the PE data would be included in one or more of Tahan’s segments. (See my analysis of Tahan’s use of “segments” in VIII.A.). Tahan’s algorithm, which it calls “MAL ID” achieves its performance “with no disassembly, Op-Code analysis, executable header analysis, unpacking nor any other preprocessing...” (Ex. 1003, 975).

171. As I explained above, these features (“static data points”) are extracted from executable files. Moreover, Tahan explains that its “paper proposes several novel methods, based on machine learning, to detect malware in executable files without any need for preprocessing, such as unpacking or disassembling.” (Ex. 1003, Abstract). A POSITA would have understood (based on this disclosure as well) that Tahan’s processes extract static data points from executable files.

172. Menahem also discloses “extracting a plurality of static data points from an executable file.” Menahem discloses extracting different features for use in ensemble and explains that with respect to combinations of machine learning approaches, “[w]hen generating multiple classifiers using different inducers and *various features (that are extracted from the same set of instances)* each classifier will perform differently” and contribute to improved performance of a super classifier that is an ensemble of classifiers. (Ex. 1004, p. 1484, emphasis added; see also *id.*, p. 1494; see also my analyses and explanations in Sections V.A. and VIII.B.). Reproduced below is Fig. 1 from Menahem illustrating the composition of an ensemble classifier or “Super Classifier”:

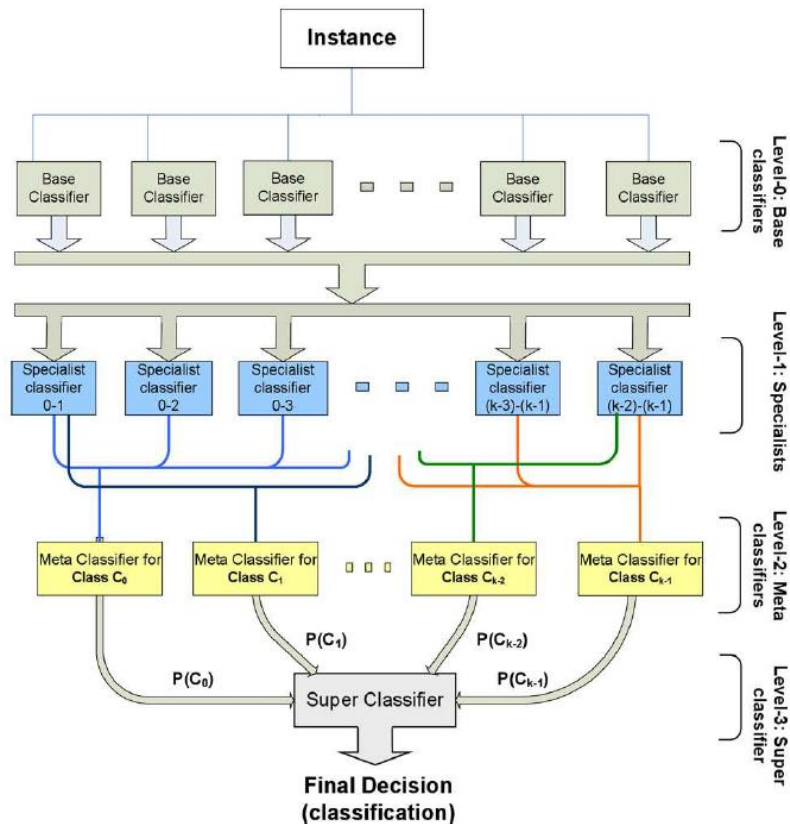


Fig. 1. Troika Architecture reveals three layers of combining classifiers; the Specialist classifiers combine the outputs of the base-classifiers, Meta-classifiers combine the Specialist classifier outputs, and finally, the Super-classifier, combines all Meta-classifier outputs. The Super-classifier is responsible for the output of Troika's prediction.

(Menahem, Fig. 1.)

173. Menahem explicitly discloses extracting (from executable files) n-grams and Portable Executable (PE) features, along with another type of feature called “function-based features”. As Menahem explains, “[w]e collected a large set of 22,735 benign files and 7690 malware that were identified as Win32 executable and DLLs. For each file **three types of features were extracted: n-grams, Portable Executable (PE) features and Function-based features as described below.**” (Ex. 1004, p. 1487, emphasis added). All three types of features disclose the claimed “static data points” for the reasons discussed above and those discussed below.

174. Menahem explains that classifiers could be trained on these different feature sets and then combined into an ensemble classifier. “In addition, in the hereby described experiment,

we evaluated different inducers on the same datasets. In future experiments we would like to evaluate the performance of inducers that are trained on different features.”² (Ex. 1004, p. 1494).

175. With reference to the PE data, Menahem taught:

We statically extracted different Portable Executable (PE) format features that represented information contained within each Win32 PE binary (EXE or DLL)... The information extracted by this tool is as follows (total of 88 attributes):

- Information extracted from the PE-header that describes the details about physical structure of a PE binary (e.g., creation\modification time, machine type and file size)
- Optional PE-header information describing the logical structure of a PE binary (e.g., linker version, section alignment, code size and debug flags)
- Import Section: which DLLs were imported and which functions from the imported DLLs that were used
- Export Section: which functions were exported (if the file being examined is a DLL)
- Resource Directory: resources used by a given file (e.g., dialogs and cursors)
- Version Information: (e.g., internal and external name of a file, version number)

(Ex. 1004, pp. 1487-1488).

176. With respect to above-mentioned “function-based features” (which are extracted from executable files), Menahem explains that “we use a j48 decision tree classifier in order to mark the beginning and ending of functions residing in the binary representation of the files.” (Ex. 1004 at p. 1488). Using these function marks, Menahem extracts the following attributes/features, which are derived from an executable file:

- Size of file
- File’s entropy value
- Total number of detected functions
- Size of the longest function that was detected
- Size of the shortest function that was detected
- Average size of detected functions
- Standard deviation of the size of detected functions

² As I explained in Section V.A., the technique used to create (or train) a classifier is sometimes called an “inducer.”

- Number of functions divided into fuzzy groups by its length in bytes (16, 24, 40, 64, 90, 128 and 256 bytes)
 - Function ratio — the proportion of number of detected functions from the overall size of the file
 - Code ratio — the proportion of the size (in bytes) of detected functions from the overall size of the file.
- (Ex. 1004, p. 1488).

177. Through the n-grams, PE features, and function-based features discussed above, Menahem discloses all four types of static data points described in the '844 Patent including numeric values, nominal values, character/binary strings, and Boolean values. For example, with respect to **numeric values**, the '844 Patent states that “file size” is an example of a numeric value static data point. ('844 Patent at 5:58-6:22 and Table 1.1). Menahem explicitly discloses extracting and using file size, as noted above for the function-based features. (Ex. 1004, pp. 1487-1488).

178. Similarly, Menahem also discloses **nominal value** static data points. The '844 Patent states that “section counts”, “DLL counts”, and “DLL characteristics” can all be nominal values. ('844 Patent at 5:58-6:22 and Table 1.1). Menahem discloses similar characteristics such as a count of total functions and the size of functions, which a POSITA would have understood as disclosing nominal value static data points. (See *id.*; see Ex. 1004, pp. 1487-1488).

179. With respect to **character and binary string** static data points, Menahem discloses numerous examples. As described above for PE header data, Menahem extracts character strings such as the internal and external name of the file. (See Ex. 1004, p. 1488). This is consistent with disclosures of the '844 Patent (e.g., at Table 1.1) that names are examples of character/binary string static data points. (See '844 Patent at 5:58-6:22 and Table 1.1). Moreover, Menahem discloses extracting binary n-gram sequences, which I explained above are character/binary string static data points. As Menahem explains, “[w]e collected a large set of 22,735 benign files and 7690 malware that were identified as Win32 executable and DLLs. For each file three types of features were

extracted: n-grams, Portable Executable (PE) features and Function-based features as described below.” (Ex. 1004, p. 1487).

180. As shown in the description of function-based features above, Menahem also discloses extracting entropy (e.g., “File’s entropy value”), which is a value associated with binary and character strings. (Ex. 1004, pp. 1487-1488). As I explained with reference to Tahan, a POSITA would have understood the entropy calculation of the bytes to be a value associated with character/binary sequences.

181. Finally, **Boolean value** status data points, which are represent “on” or “off”, “yes” or “no”, or “true” or “false”, are also disclosed by Menahem. The ’844 Patent gives, as an example of a Boolean value static data point, the existence of some element or another, or the identification of an anomaly. (’844 Patent at 5:58-6:22 and Table 1.1). Menahem, for example, discloses identifying and extracting “debug flags” from the PE executable. (Ex. 1004, p. 1487). As I explained above in Section V.B., a POSITA would have understood that “flags” are configuration elements that are either on or off (e.g., a Boolean value). (See my explanations above in Section V.B.).

182. I have already explained in section IX.A. why a POSITA would be motivated to combine Tahan and Menahem, and I incorporate this analysis by reference. As I explained in this analysis, a POSITA would have recognized that combining the different types of classifiers into an ensemble can produce a better classifier. As Menahem explains, “The main idea of an ensemble methodology is to combine a set of classifiers in order to obtain a better composite global classifier. Ensemble methodology imitates our second nature to seek several opinions before making any crucial decision. We weigh the individual opinions, and combine them to reach a final decision.” (Ex. 1004 at p. 1484). Substituting an ensemble classifier for a single base classifier was known

in the art, as discussed in prior art and in the '844 Patent itself. (See, e.g., '844 Patent at 9:41-67; see also my explanations in Section VIII.B.).

183. Menahem also teaches that it was well known in the art that the effectiveness of a classifier was determined in part by its features (that disclose the claimed “static data points”): “By its nature, the measured performance (e.g., accuracy) of a machine learning classifier is influenced by... the types of features that are used to represent the instances.” (Ex. 1004 at 1484). Thus, with respect to the ensemble, Menahem discloses that the “combined” features for the super classifier/ensemble classifier could be composed of a combination of different features, each for its own classifier. “In future experiments we would like to evaluate the performance of inducers that are trained on different features.”³ (Ex. 1004, p. 1494; see also my explanations in Section VIII.B.). Menahem identifies that this was desirable. “When generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently” and contribute to improved performance of a super classifier that is an ensemble of classifiers. (Ex. 1004, p. 1484; see also *id.* at p. 1494; see also my analyses and explanations in Section VIII.B.).

184. A POSITA would have looked to a reference like Menahem, which also combines machine-learning algorithms, in order to obtain these advantages. Menahem discloses an “ensemble” combination of machine learning approaches/classifiers: “[i]n order to benefit from multiple different classifiers, and exploit their strengths we suggest using an ensemble method that will combine the results of the individual classifiers into one final result to achieve overall higher detection accuracy...” (Ex. 1004, Abstract). In doing so, a POSITA would have taken advantage

³ As I explained in Section V.A., the technique used to create (or train) a classifier is sometimes called an “inducer.”

of Menahem's and Tahan's wide array of extracted features ("static data points") and the classifiers trained to use them in order to improve the accuracy of the classifier.

185. Thus, Tahan alone or in combination with Menahem discloses this element of the limitation.

"without decrypting or unpacking the executable file"

186. This is the second element, which is disclosed by Tahan.

187. For example, Tahan, teaches that techniques discussed in Tahan (which include, *inter alia*, extracting features (static data points) from executable files) can be performed "without any need for preprocessing, such as unpacking or disassembling." Tahan explains, "This paper proposes several novel methods, based on machine learning, to detect malware in executable files ***without any need for preprocessing, such as unpacking or disassembling.***" (Ex. 1003, Abstract, emphasis added; see also p. 975). In my opinion, a POSITA would have understood, based on the disclosure "without any need for preprocessing, such as unpacking or disassembling," that an executable file (to the extent it is encrypted or packed) does not have to be decrypted or unpacked for the techniques (e.g., including extraction of static data points from an executable file) to be performed. (*See id.*). Moreover, a POSITA would have understood a decrypting or unpacking process (of an executable file) to be a type of preprocessing (of an executable file), and thus would have understood that static data points can be extracted from an executable file without decrypting or unpacking the executable file, based on Tahan's disclosures. To the extent Patent Owner incorrectly argues that this is not disclosed by Tahan, a POSITA would have been motivated to implement Tahan's techniques (including extraction of features from an executable file) "without decrypting or unpacking the executable file," based on Tahan's disclosures. (*See id.*)

188. Moreover, Tahan explains that its techniques and processes work with executable files in general or “in the Wild.” (See *id.*; see also *id.*, p. 958 (“The input for the Mal-ID method is an unclassified executable file of any size.”), p. 975 (“Based on the results presented so far, we hypothesize that the proposed Mal-ID method performs well in a mixed environment where both obfuscated (including compressed or encrypted) and **plain executable files** exist. In this sense, we referred to **malware as they are found ‘in the Wild.’**”), emphasis added). A POSITA would have understood that executable files in general or “in the Wild” may include executable files that are not encrypted or packed. This is consistent with the ’844 Patent’s explanation regarding what may be encompassed by executable files “in the wild”:

The present disclosure also provides for the creation of a training set that adaptively learns what static data points may indicate the existence of malicious code using training data that contains a statistically significant number of both encrypted and non-encrypted examples of malicious or unwanted **executable files**, among other information. By collecting a large set of training examples as they appear as publicly available and distributed over network connection (e.g., “**in the wild**”), the present disclosure ensures that its adaptive learning processing is robust enough to comprise sufficient representation of features (and distributions) from **files that may be encrypted, not-encrypted, compressed, and uncompressed**, among other examples.

(’844 Patent at 2:32-45, emphasis added).

189. A POSITA would have understood that uncompressed files are unpacked files. Thus, based on Tahan’s disclosure, a POSITA would have understood that features / static data points could be extracted from such non-encrypted / unpacked executable files without decrypting or unpacking the file. (See *id.*)

190. Thus, Tahan discloses this element.

“wherein the plurality of static data points represent predefined character strings in the executable file”

191. This is the third and final element of this limitation. Tahan combined with Menahem discloses this element.

192. As explained above, Tahan alone or combined with Menahem discloses extracting from executable files all of the different types of static data points described by the '844 Patent (e.g., numeric values, nominal values, strings/byte sequences, Boolean values).

193. Tahan discloses extracting “static data points” that “represent predefined character strings in the executable file”. For example, Tahan discloses extracting segments (sequences of binary data) that could represent an entire code line. “In many cases 3- to 8-byte n-gram cannot represent even one line of code composed with a high level language. In fact, we showed in a previous paper (Tahan et al., 2010) that an n-gram should be at least 64 bytes long to uniquely identify a malware... Note that, when a segment represents at ***least one line of code developed using a high level language***; it can address the second deficiency of using short features that may be meaningless when considered alone.” (Ex. 1003, p. 952, emphasis added). Thus, Tahan teaches that, even though the extracted features (or static data points) are just n-gram sequences from within the file, the sequences represent data such as lines of code and other strings.

194. It should be noted that Tahan’s “segments” are intended to represent meaningful portions⁴ of the executable file. As I will discuss below with reference to limitation 1[c], Tahan teaches the use of a Common Function Library (CFL), which “contains data structures constructed

⁴ U.S. Patent No. 9,465,940 (“Wojnowicz”), cited by the Examiner during prosecution, contains the concept of “non-overlapping file chunks of fixed length”, which Patent Owner argued “do not represent predefined character strings. Instead, each fixed-length file chunk is, itself, a random (e.g. undefined) character string having no initial significance to the system of Wojnowicz.” (Ex. 1007, pp. 155-156). Tahan’s “segments”, unlike Wojnowicz’s “file chunks of fixed length”, intend to represent not “a random (e.g., undefined) character string having no initial significance to the system” (*id.*) but a meaningful portion of an executable file (e.g., “at least one line of code” (Ex. 1003, p. 952)).

from benign files”, and a Threat Function Library (TFL), which “contains data structures constructed from malware without segments identified as benign”. (Ex. 1003, p. 955). A POSITA would have understood, from the use of terms like “data structures” and “library”, that these libraries are intended to represent meaningful (i.e., not merely random) segments from known benign and known malicious files. Segments (static data points) extracted from the executable file are then compared with segments from the CFL and TFL and used in the classification process. (See Ex. 1003, p. 958; see also my analysis of limitation 1[c] hereinbelow). A POSITA would have understood that such comparison is intended to identify meaningful (not merely random) segments from the executable file (e.g., representing a line of code), especially in light of Tahan teaching that “when a segment represents at ***least one line of code developed using a high level language***; it can address the second deficiency of using short features that may be meaningless when considered alone.” (Ex. 1003, p. 952, emphasis added). Thus, Tahan discloses extracting “static data points” that “represent predefined character strings in the executable file”.

195. In another example, as I discussed with reference to the first element of limitation 1[b] (“extracting a plurality of static data points from an executable file”), Tahan discloses that it was common in the art to use PE header fields such as debug flags, CPU type, etc. (Ex. 1003 at p. 950). Such PE header fields are predefined character strings or static data points that “represent” predefined character strings because such fields are predefined fields (e.g., recognized by the system) with character strings or values represented by character strings.

196. Similarly, Menahem also discloses “wherein the plurality of static data points represent predefined character strings in the executable file”. Most, if not all, of Menahem’s PE header features and function-based data are either predefined character strings or static data points

or that “represent” predefined character strings. PE header fields are predefined character strings or static data points that “represent” predefined character strings for the reason discussed above.

197. For example, “machine type”, “linker version”, and “debug flags” are all predefined character strings. Menahem also discusses imported functions and exported functions. Menahem lists the following:

We statically extracted different Portable Executable (PE) format features that represented information contained within each Win32 PE binary (EXE or DLL)... The information extracted by this tool is as follows (total of 88 attributes):

- Information extracted from the PE-header that describes the details about physical structure of a PE binary (e.g., creation\modification time, machine type and file size)
- Optional PE-header information describing the logical structure of a PE binary (e.g., linker version, section alignment, code size and debug flags)
- Import Section: which DLLs were imported and which functions from the imported DLLs that were used
- Export Section: which functions were exported (if the file being examined is a DLL)
- Resource Directory: resources used by a given file (e.g., dialogs and cursors)
- Version Information: (e.g., internal and external name of a file, version number)

(Ex. 1004, pp. 1487-1488).

198. These are values associated with predefined strings identifying import and export sections of the PE header.

199. As discussed above, with respect to “function-based features,” Menahem explains that “we use a j48 decision tree classifier in order to mark the beginning and ending of functions residing in the binary representation of the files.” (Ex. 1004 at p. 1488). Using these function marks, Menahem extracts the following attributes/features, which are derived from an executable file:

- Size of file
- File’s entropy value

- Total number of detected functions
- Size of the longest function that was detected
- Size of the shortest function that was detected
- Average size of detected functions
- Standard deviation of the size of detected functions
- Number of functions divided into fuzzy groups by its length in bytes (16, 24, 40, 64, 90, 128 and 256 bytes)
- Function ratio — the proportion of number of detected functions from the overall size of the file
- Code ratio — the proportion of the size (in bytes) of detected functions from the overall size of the file.

(Ex. 1004, p. 1488).

200. I have already explained in section IX.A. and above for this limitation why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference. As I explained in this analysis, a POSITA would have recognized that combining the different types of classifiers into an ensemble can produce a better classifier. Menahem also explains that using different kinds of data (e.g., features) in an ensemble is one way of creating/training an ensemble classifier, and that the value (e.g., performance) of a classifier is based on (in part) the features chosen.

201. As explained by Menahem, “The main idea of an ensemble methodology is to combine a set of classifiers in order to obtain a better composite global classifier. Ensemble methodology imitates our second nature to seek several opinions before making any crucial decision. We weigh the individual opinions, and combine them to reach a final decision.” (Ex. 1004 at p. 1484). Substituting an ensemble classifier for a single base classifier was known in the art, as discussed in prior art and in the ’844 Patent itself. (See, e.g., ’844 Patent at 9:41-67; see also my explanations in Section VIII.B.). In doing so, a POSITA would have taken advantage of Menahem’s wide array of features and the classifiers trained to use them in order to improve the accuracy of the super classifier/ensemble classifier.

202. Thus, Tahan in combination with Menahem discloses this element.

203. Thus, I conclude that Tahan in view of Menahem discloses limitation 1[b].

3. Limitation 1[c]

204. Limitation 1[c] of the '844 Patent reads:

generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range; and;

205. Tahan in view of Menahem discloses limitation 1[c] of the '844 Patent. I have decomposed this limitation into four elements.

“generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points”

206. This is the first element, which Tahan, alone or in combination with Menahem, discloses.

207. Tahan discloses “generating a feature vector from the plurality of static data points.” For example, Tahan explains that after features / static data points are extracted (as discussed above for limitation 1[b]), a feature vector containing the extracted features (“a **vector** containing the **features** extracted”) is created/generated:

The techniques use three distinct stages:

1. Feature Extraction for file representation—The result of the feature extraction phase is a **vector** containing the **features** extracted.

...

2. Feature Selection (or feature reduction)—During this phase the **vector created in phase 1** is evaluated and redundant and irrelevant features are discarded.

...

3. The last phase is creating a classifier using the reduced features vector created in phase 2 and a classification technique.
(Ex. 1003, pp. 950-951, emphasis added).

208. Tahan explains that the result of the first stage/phase is that a feature vector containing features / static data points is generated from the extracted features / static data points. It also explains that this feature vector generated in the first phase (“**vector created in phase 1**”) is used in subsequent stages/phases. (See *id.*) Based on this disclosure, a POSITA would have understood that Tahan’s processes “generat[e] a feature vector from the plurality of [extracted] static data points.”

209. Tahan also discloses “generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points.” Tahan explains that the processes described in its paper are performed by a machine learning mechanism/module/application. For example, Tahan discloses machine learning mechanisms/modules/applications (for example, ones referred to as Mal-ID) that use machine learning to classify executable files and detect malware. (See, e.g., Ex. 1003, Abstract (“This paper proposes several novel methods, based on **machine learning, to detect malware in executable files** without any need for preprocessing, such as unpacking or disassembling. The basic method (Mal-ID) is a new static (form-based) analysis methodology that uses common segment analysis in order to detect malware files. . . . In addition, the results indicated that by

extracting meaningful features, it is sufficient to employ one simple detection rule for *classifying executable files.*”), emphasis added). Therefore, and for the reasons provided below, a POSITA would have understood that machine learning mechanisms/modules/applications in Tahan disclose classifiers. (See also my analysis for limitation 1[d].)

210. A POSITA would understand that “a classifier” in the claim language requiring “generating a feature vector... using a classifier” encompasses the “classifier component” (or classifier module) as I discussed in my overview of the ’844 Patent above. (See Section VI.A.). As I noted above, a POSITA would have recognized that the specification sometimes uses the term “classifier” to mean “classifier component.” (See *id.*). I also explained that a POSITA would have known that this was common usage in the art, and provided examples from computer programming libraries. (See *id.*). For example, the classifier software in the Scikit-Learn library is organized into modules that perform operations using APIs (Application Programming Interfaces). (See Ex. 1011 at p. 4). Through APIs, a programmer can configure the classifier, perform feature selection, train the model, and so forth. Within Scikit-Learn, all of these APIs are grouped **together into a single module for each type of classifier**. (See *id.*). Specifically, this grouped classifier module includes all of the elements required to perform classification including feature extraction, training, and so forth. (See *id.* at p. 7). A POSITA would have understood that the above-discussed grouped (classifier) module in Scikit-Learn (and the above-discussed learning mechanisms/modules/applications in Tahan) are substantially similar in concept to the above-discussed broader “classifier component” (e.g., “learning classifier component”) in the ’844 Patent. (See Section VI.A.).

211. The ’844 Patent specification sometimes uses the term “classifier” to mean the broader “classifier component” (as well as a classifier algorithm, which may be part of the broader

classifier component). A good example of the use of “classifier” to refer to both the “classifier component” as well as the “classifier” (algorithm) is illustrated in the following citation from the specification: “In one example, operation 306 comprises generating at least one feature vector from the plurality of static data points extracted **using** the learning classifier... In order to generate the feature vector **for** the learning classifier, data may be parsed and encoded for machine learning processing.” (’844 Patent at 12:48-53, emphasis added). A POSITA would have recognized that the term “classifier” (e.g., in the specification) sometimes means the broader “classifier component.” For example, in the quote above, a POSITA would have understood that the “learning classifier” in the phrase “generating at least one feature vector from the plurality of static data points extracted using the learning classifier” would be a broader classifier component, which performs certain tasks such as extracting static data points and “generat[ing] the feature vector **for the learning classifier,**” wherein this latter instance of “learning classifier” refers more specifically to a classifier algorithm (e.g., in the learning classifier component). (See *id.*). A POSITA would have understood that the learning classifier component extracts features, and that those features are used in training the classifier algorithm (which may be part of the broader “classifier component”) and/or in classifying an unclassified object.

212. Thus, a POSITA would have understood that machine learning mechanisms/modules/applications in Tahan disclose classifiers. (See also my analysis for limitation 1[d].) Such mechanisms/modules/applications perform the processes discussed in Tahan (including extraction of static data points and generation of a feature vector from the extracted static data points). (See generally, Ex. 1003). For example, as discussed in Sections VIII.A and IX.A. and for Claim 1, Tahan’s classifier mechanism/module/application such as Mal-

ID performs such processes. Thus, Tahan discloses “generating a feature vector from the plurality of static data points using a classifier.”

213. Moreover, Tahan also discloses that such a classifier is “trained to classify the plurality of static data points.” For example, as I discussed above (e.g., for Sections V.A., VIII.A.), Tahan’s classifier is a trained classifier. I incorporate such analyses by reference. Moreover, Tahan discloses that such a classifier is “trained to classify the plurality of static data points.” For example, as I discussed above and for Section VIII.A., Tahan’s classifiers (such as Mal-ID) perform common segments analysis where segments (which I explained above for VIII.A. and limitation 1[b] disclose features / static data points) can be identified or classified as malicious or benign (e.g., a classifier is “trained to classify the plurality of static data points” (e.g., as malicious or benign)). (I incorporate by reference my analysis for Section VIII.A. and limitation 1[b].)

214. For example, as I explained above, Tahan discloses analyzing segments (static data points) in order to determine which segments are common to benign files and malicious files. The concept is related to features (static data points) in that features that show up in both malware and benign files are preemptively removed, reducing the feature set. Tahan explains that “[a]ll applications... that were developed using a higher level development platforms [sic]... contain common code and resources that originate from common code and resource libraries. Since most malware are also made of the same common building blocks, we believe it would be reasonable to discard the parts of a malware that are common to all kinds of software, leaving only the parts that are unique to the malware.” (Ex. 1003 at p. 952).

215. Once the common segments are removed, Tahan identifies segments that are common to the malware (based on a training set). “In other words, when given an unclassified file, the first step would be to detect the file segments that originated from the development

platform or from a benign third party library (termed here the Common Function) and then disregard those segments. Finally, the remaining segments would be compared to determine their degree of resemblance to a collection of known malwares. If the resemblance measure satisfies a predetermined threshold or rule then the file can be classified as malware.” (Ex. 1003 at p. 955). Tahan collects the “common” (benign) segments into a Common Function Library (CFL) and segments that only appear in malware into a Threat Function Library (TFL). (*Id.*).

216. Sequences/segments (static data points) in the CFL and TFL are then pre-defined binary sequences/segments used by the classifier and against which other files are compared. Tahan explains that segments should be discarded, for example, if “it may appear in benign files.” (Ex. 1003 at p. 958). On the other hand, “segments that actually reside in the malware are left in the segment collection.” (Ex. 1003 at p. 959).

217. The following figure from Tahan illustrates combining these various elements together. During the setup phase, the system identifies a common function library (CFL) that identifies sequences/segments that are in benign software and the threats function library (TFL) that identifies sequences/segments that are only in malware (based on training data). During the detection phase, the potential malware is broken into segments (static data points). Segments are further categorized/classified by whether they are in the CFL (e.g., containing and indicating benign segments (static data points)) or TFL (e.g., containing and indicating harmful segments (static data points)). (See also Ex. 1003, p. 955 (“The CFL repository is constructed from benign files and the TFL repository is constructed from malware files. These repositories are later used to derive the meta-features—as described in Section 2.2. Note that in the proposed algorithm, we are calculating the distribution of 3-grams within each file and across files, to make sure that a 3-gram

belongs to the examined segment and thus associate the segment [static data point] to either benign (CFL) or malware (TFL).”).)

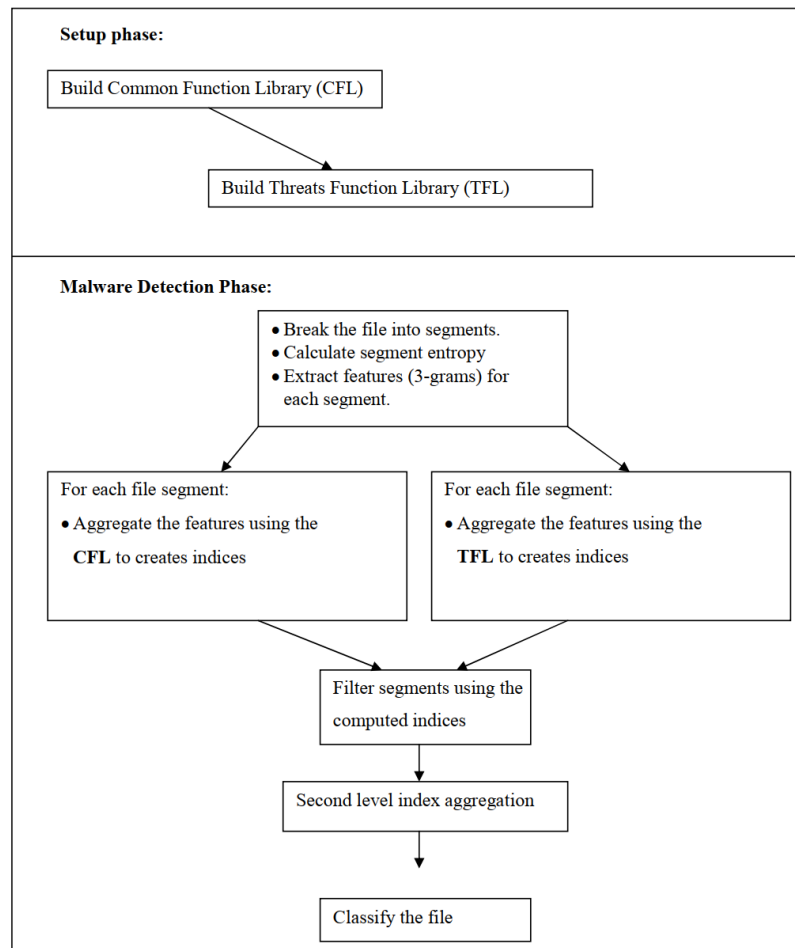


Figure 1: The Mal-ID method for detecting new malware applications.

218. Thus, Tahan’s classifiers (including Mal-ID) are trained to classify segments (static data points) as malicious or benign. Thus, Tahan discloses “a classifier trained to classify the plurality of static data points.”

219. Also, Tahan and Menahem disclose “a classifier trained to classify the plurality of static data points” (e.g., classify static data points into categories or data types) for the reasons discussed for Claims 2 and 4 below. I incorporate by reference my analyses for Claims 2 and 4.

220. In addition, as I explained in my overview of the reference, Tahan teaches that feature vectors could be improved by the use of “meta-features” extracted from the n-grams, and the exclusion of data found commonly between both benign and malicious software. “The basic method (Mal-ID) is a new static (form-based) analysis methodology that uses common segment analysis in order to detect malware files. By using common segment analysis, Mal-ID is able to discard malware parts that originate from benign code. In addition, Mal-ID uses a new kind of feature, termed meta-feature, to better capture the properties of the analyzed segments.” (Ex. 1003 at p. 949; see also, e.g., Tahan, pp. 952, 954, 955, 957-958).

221. Beyond the specifics taught by Tahan for its own improvements to the art, Tahan also discloses that this basic approach was well known for this general solution of applying machine learning to classifying malware.

The techniques use three distinct stages:

1. Feature Extraction for file representation—The result of the feature extraction phase is a vector containing the features extracted

...

2. ***Feature Selection (or feature reduction)***—***During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded.*** Feature selection has many benefits including: improving the performance of learning modules by reducing the number of computations and as a result the learning speed; enhancing generalization capability; improving the interpretability of a model, etc. Feature selection can be done using a wrapper approach or a correlation-based filter approach (Mitchell, 1997). Typically, the filter approach is faster than the wrapper approach and is used when many features exist. The filter approach uses a measure to quantify the correlation of each feature, or a combination of features, to a class. The overall expected contribution to the classification is calculated and selection is done according to the highest value. The feature selection measure can be calculated using many techniques, such as gain ratio (GR); information-gain (IG); Fisher score ranking technique (Golub et al., 1999) and hierarchical feature selection (Henchiri and Japkowicz, 2006).

3. The last phase is creating a classifier using the reduced features vector created in phase 2 and a classification technique.
(Ex. 1003, pp. 950-951, emphasis added).

222. A POSITA would have recognized that Tahan is describing an overall approach to the machine learning process that underlies the rest of Tahan's disclosures, in which Tahan builds on the approach. The '844 Patent specification teaches that the "learning classifier component" creates or builds the classifier:

As an example, the learning classifier 104 applies programming operations or machine learning processing to evaluate static data points extracted from a file to be analyzed. In doing so, the learning classifier component 104 builds a learning classifier based on information from the knowledge component 102 including the extracted static data points of a file and the information used to train/re-train the learning classifier (e.g., static data on the robust collection of executable files including variety of files types of malicious executable files, benign executable files, and potentially unwanted executable files).
('844 Patent at 7:46-56).

223. A POSITA would have understood that because Tahan's approach uses machine learning processes with extracted features ("static data points") to produce a classifier, it describes the same concept as the '844 Patent of "build[ing] a classifier". Tahan's description of the approach teaches feature selection and the creation of a "features vector". As I explained with reference to limitation 1[b], the features represent "static data points". Therefore, a POSITA would have recognized that the classifier so produced teaches the claimed "classifier trained to classify the plurality of static data points".

224. Menahem also discusses generating a feature vector using the learning classifier component. "We developed a tool that extracts n-grams from the binary representation of a file. The tool parsed the files using a sliding window producing n-grams of varying lengths (denoted by n)... we [also] statically extracted different Portable Executable (PE) format features that represented information contained within each Win32 PE binary (EXE or DLL)... and we applied

the Gain Ratio feature selection method (Mitchell, 1997) on the n-gram-based and PE datasets.” (Ex. 1004, pp. 1487-1488).

225. Moreover, Menahem discloses creating a “super” classifier or ensemble classifier by combining diverse component/base classifiers (e.g., classifiers that have been trained on unique features): “In future experiments we would like to evaluate the performance of inducers that are trained on different features.” (Ex. 1004, p. 1494). Menahem identified that this was desirable. “When generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently” and contribute to improved performance of a super classifier that is an ensemble of classifiers. (Ex. 1004, p. 1484, emphasis added; see also *id.*, p. 1494; see also my analyses and explanations in Sections V.A. and VIII.B.).

226. Although Tahan discloses this element, a POSITA would have also looked to Menahem for additional details. I have already explained in section IX.A. why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference.

227. Also, to the extent “a classifier trained to classify the plurality of static data points . . .” of this limitation is interpreted to mean that such a classifier classifies (or is trained to classify) the extracted static data points “according to a type of data extracted from the executable file” (e.g., Claim 2) or “into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values” (e.g., Claim 4), the Tahan-Menahem method/system discloses or renders obvious such classification processes/techniques (for the reasons discussed below for Claims 2 and 4). I incorporate by reference my analyses for Claims 2 and 4. Moreover, as discussed above, Tahan-Menahem’s machine learning mechanisms/modules/applications (e.g., similar to the broader “classifier component”) disclose “a classifier” recited by this limitation, and

such a classifier performs the processes of the Tahan-Menahem method/system. Thus, a POSITA would have understood that such a classifier performs the method/system's process of classifying the plurality of static data points—e.g., “according to a type of data extracted from the executable file” or “into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values”, or at least would have been motivated to configure it in such a manner. Thus, the Tahan-Menahem method/system discloses or renders obvious “a classifier trained to classify the plurality of static data points” and this limitation under the above-referenced interpretation as well.

228. Thus, Tahan alone or in combination with Menahem discloses this element.

“based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files,”

229. This is the second element. Tahan discloses using training sets (“collection of data”) of known benign, known malicious, and known unwanted executables to train a classifier. Tahan uses “programs located under Windows XP program files folders” for the benign samples and malware acquired from an anti-virus company for the malicious or unwanted software. (Ex. 1003, p. 955). Tahan includes, in its categorization of malware, “spyware”, which a POSITA would have understood is known unwanted software (including unwanted executable files). (Ex. 1003, pp. 949-950). For example, Tahan explains that “Spyware [is] a software **installed** on a computer system **without the user's knowledge** to collect information about the user.” (*Id.*, p. 950, emphasis added). This is consistent with the '844 patent, which explains that “potentially unwanted executable files are files that may be installed on system 100 that may not be malicious, but a user of the device may not want such a file to execute and/or **a file that is executed/installed unknowingly to a user.**” ('844 Patent, 3:66-4:3, emphasis added). Moreover, a spyware is

unwanted because a user would not “want such a file to execute” since it would “collect information about the user.” (*See id.*)

230. Tahan teaches extracting n-grams from known benign and known malicious files to create two data structures: a “Common Function Library” (CFL) containing n-grams from known benign files and a “Threat Function Library” (TFL) containing n-grams from known malicious files (see Ex. 1003, p. 955). Then, these data structures are applied to the training set data to produce a classifier:

1. First, the Mal-ID basic method is used to construct the CFL and TFL. This stage is performed only once before the file classification starts.

2. Next, zero patch each malware in the training set as follows: Iterate over all of the file segments and perform common segment analysis to detect the segments that appear in the CFL. The benign segments (the segments that appear in the CFL) are zero patched in an attempt to reduce the number of n-gram that are clearly not relevant for detecting segments that appear only in malware. The end result is a new file with the same length that has zeros in the benign segments.

3. Finally, ***construct a classification model*** using Rotation Forest using J48 decision tree as base classifier. ***The patched malware collection and the unchanged benign file collection are used for training.*** (Ex. 1003, p. 961, emphasis added).

231. A POSITA would have understood that Tahan teaches the claimed “collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files”, and that this data set is used to train the claimed “classifier trained to classify the plurality of static data points”, described in the preceding element. Also, a POSITA would have understood that the classifier is trained based on “collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files,” based on POSITA’s knowledge and for the reasons discussed above. (See Sections V.A., VIII.A.).

232. Menahem’s training data set also collected executable files whose benign/malware status was known beforehand: “[w]e collected a large set of **22,735 benign files and 7690 malware** that were identified as Win32 executable and DLLs. For each file three types of features were extracted: n-grams, Portable Executable (PE) features and Function-based features as described bellow [sic].” (Ex. 1004 at p. 1487, emphasis added). Menahem further discloses using this data set to train classifiers (which it calls “inducers”⁵): “After performing **rigorous experiments in which we evaluated several inducers on the 192 different n-gram datasets**, we chose the top 300 5-grams with TF-IDF values, top 300 6-grams with TF-IDF and top 300 6-grams, with TF datasets providing the best results.” (Ex. 1004, p.1487, emphasis added). A POSITA would have understood that Menahem’s language “evaluated several inducers on the...datasets” teaches training a classifier using the datasets in order to measure the effectiveness of the classifier. (See, e.g., Ex. 1004, p. 1488).

233. A POSITA would have recognized that both Tahan and Menahem teach training a classifier (the claimed “classifier trained to classify the plurality of static data points”) using a data set, and that such data sets represent the same concept as the claimed “collection of data comprising known malicious executable files, known benign executable files, and known unwanted files”. Also, a POSITA would have understood that the classifier is trained based on “collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files,” based on POSITA’s knowledge and for the reasons discussed above. (See Sections V.A., VIII.A., VIII.B.).

⁵ As I explained in Section V.A., the technique used to create (or train) a classifier is sometimes called an “inducer.”

234. I have already explained in section IX.A. why a POSITA would be motivated to combine Tahan and Menahem, and I incorporate this analysis by reference.

235. Thus, Tahan alone or in combination with Menahem discloses this element.

“wherein the collection of data comprises at least a portion of the plurality of static data points,”

236. This is the third element, which is disclosed by Tahan alone or in view of Menahem.

237. As I explained above (e.g., for Technical Background (Sections V.A-B), Section VIII.A-B, and limitation 1[b]), features (static data points) are extracted from executable files. (See my analyses and explanations above for Sections V.A-B, XIII.A-B and for limitation 1[b]). Thus, features (static data points) are part of executable files. (See *id.*) As discussed above, the claimed “collection of data” (as disclosed by Tahan and Menahem) comprises executable files (e.g., from which features (static data points) are or can be extracted). (See *id.*) Thus, a POSITA would have understood that “the collection of data comprises at least a portion of the plurality of static data points.” (See *id.*)

238. Moreover, as described with respect to limitation 1[b], static data points are extracted from both training data sets (“collection of data”) (used to train a classifier) and from a file (e.g., a potentially malicious executable file) to be evaluated. A classifier trained on static data points extracted from a training data set may be used to classify the file (to be evaluated) using the static data points extracted from the file. (See my explanation in Technical Background Section V.A.). That is, a POSITA would have understood that static data points are extracted from training data sets when training a classifier, and static data points are also extracted from a file to be evaluated when using the classifier to classify the file. Thus, my analysis with respect to extracting features (including static data points) for classification/training (from limitation 1[b] and Section

V.A.) applies equally to this element. (I incorporate my analysis of limitation 1[b] and Section V.A. into this section by reference.)

239. A POSITA would have understood that because certain features (static data points) are extracted from executable files (for one example, certain PE header field features are extracted from Portable Executable files, which (as discussed above) are also in the “collection of data”), there are some overlaps between features extracted from a file to be evaluated and features in the “collection of data” or features present in the executable files in the “collection of data.” (See *id.*; see also Section V.B.; see also my analysis for the element above (e.g., discussing that Menahem’s training data includes features / static data points)). Thus, a POSITA would have understood that Tahan and Menahem disclose “wherein the collection of data comprises at least a portion of the plurality of static data points.”

240. Thus, Tahan in combination with Menahem discloses this element.

“and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range;”

241. This is the fourth element. Tahan discloses this element. Tahan emphasizes that turning on or off features is an essential component of the process.

242. Tahan describes the feature selection/reduction process, which a POSITA would have understood includes selectively turning on or off features of a feature vector, as a common technique when using machine learning to classify a file and to detect malware.

In addition, in recent years many researchers have been using machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files.

The techniques use three distinct stages:

...

2. **Feature Selection (or feature reduction)**—During this phase the vector created in phase 1 is evaluated and **redundant and irrelevant features are discarded. Feature selection has many benefits including: improving the performance of learning modules by reducing the number of computations and as a result the learning speed**; enhancing generalization capability; improving the interpretability of a model, etc. Feature selection can be done using a wrapper approach or a correlation-based filter approach (Mitchell, 1997). Typically, the filter approach is faster than the wrapper approach and is used when many features exist. **The filter approach uses a measure to quantify the correlation of each feature, or a combination of features, to a class.** The overall expected contribution to the classification is calculated and selection is done according to the highest value. **The feature selection measure can be calculated using many techniques, such as gain ratio (GR); information-gain (IG); Fisher score ranking technique (Golub et al., 1999) and hierarchical feature selection (Henchiri and Japkowicz, 2006).**” (Ex. 1003, pp. 950-951, emphasis added).

243. Thus, not only does Tahan disclose this limitation, but it makes it clear that it was a common practice and would have been known to a POSITA.

244. For example, Tahan explains (in the above quote), “Feature Selection (or feature reduction)—During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded.” (See *id.*) As discussed above, phase 1 refers to the processes of extracting features (static data points) from an executable file and generating a feature vector from the extracted features (static data points). (See, e.g., Ex. 1003, p. 950 (“1. **Feature Extraction** for file representation—**The result of the feature extraction phase is a vector containing the features extracted.**”); see also my analyses for Sections V.A-B, Sections VIII.A-B, and for limitations 1[b], 1[c].) As discussed above, a POSITA would have understood that “a vector containing the features extracted” refers to a feature vector with extracted features (static data points). (See *id.*) Thus, a POSITA would have understood that “vector” in the quote above (“Feature Selection (or feature reduction)—During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded”) also refers to a feature vector (e.g.,

feature vector containing extracted features (static data points)). (See *id.*; see also Ex. 1003, p. 951). And, based on the phrases “Feature Selection (or feature reduction)” and “the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded,” a POSITA would have understood that “one or more features of the feature vector are selectively turned on or off” (e.g., if the feature selection (feature reduction) process discards a feature of a feature vector, the feature is considered turned off (e.g., and would have less impact on subsequent processes such as evaluating a feature vector to determine if a file is harmful), and if a feature of a feature vector is kept, it is considered turned on (e.g., and would have greater impact on subsequent processes such as evaluating a feature vector to determine if a file is harmful)).

245. Moreover, Tahan discloses turning on or off features “based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range.” For example, Tahan discloses an algorithm to “classify a file F as benign or as malware” (Ex. 1003, p. 958). It does so by checking whether or not certain values (e.g., entropy values) associated with segments (which, as I explained above for limitation 1[b], correspond to n-grams, which are features (static data points) extracted from executable files) are within a certain predetermined range (e.g., between EntropyLow and EntropyHigh), and excluding/discarding certain segments if the entropy values are not in the predetermined range: “In this line we check if the entropy is smaller than EntropyLow threshold or entropy is larger than EntropyHigh. Is [sic] so then **discard the segment** and continue segment iteration.” (Ex. 1003, p. 958, emphasis added; see also my analyses and explanations for Sections VIII.A and limitations 1[b] and 1[c]). As I discussed above, Tahan discloses generating a feature vector from features (static data points) extracted from an executable file. (See, e.g., my analysis above for this

limitation 1[c]). Thus, a POSITA would have understood that the segments are part of feature vectors (e.g., features of feature vectors). (*See id.*)

246. A POSITA would have recognized that discarding a segment as Tahan describes lessens the impact that segment will have on the classification process. A POSITA would have understood this technique to be consistent with the feature selection (feature reduction) process discussed above. (*See id.*; see also Ex. 1003, p. 951 (“**2. Feature Selection (or feature reduction)**—During this phase the **vector** created in phase 1 is evaluated and **redundant and irrelevant features are discarded.**”), emphasis added). A POSITA would have understood this technique to teach the claimed “one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range”.

247. Moreover, as I discussed above for limitation 1[b], entropy is a static data point (which is consistent with disclosures of the ’844 patent). (See my analysis for limitation 1[b]). Thus, turning on or off “one or more features of the feature vector . . . based on whether” an entropy value “is within a predetermined range” discloses this claim element.

248. In another example, Tahan discloses using feature sorting to turn on and off features based on values of features (static data points). In this scenario, the claimed “predetermined range” is disclosed by a predetermined range within the ranking disclosed by Tahan. That is, a score is computed for each feature (static data point) based on its value (e.g., to determine “[t]he overall expected contribution to the classification” process (Ex. 1003, p. 951)), and only features within a predetermined ranking are enabled (e.g., turned on to be used in subsequent processes such as evaluating a feature vector to determine if a file is harmful). Thus, features are selectively turned

on or off if they are within (or outside) a predetermined range (and those features that are turned off will have less impact on the later classification process).

249. Tahan describes this process this way:

The filter approach uses a measure to quantify the correlation of each feature, or a combination of features, to a class. The overall expected contribution to the classification is calculated and ***selection is done according to the highest value***. The ***feature selection*** measure can be calculated using many techniques, such as ***gain ratio (GR)***; information-gain (IG); Fisher score ranking technique (Golub et al., 1999) and hierarchical feature selection (Henchiri and Japkowicz, 2006). (Ex. 1003, p. 951, emphasis added).

250. Tahan discloses using the “Gain Ratio feature selection” process. “We will refer to our variation of Kolter and Maloof method as GR500BDT as it uses ***Gain Ratio feature selection***, 500 4-grams, and Boosted Decision Tree classifier.” (Ex. 1003, p. 961, emphasis added).

251. Menahem also discloses this element. Menahem, for example, provides motivation for narrowing the features used for classification: “In ML applications, the large number of features in many domains presents a huge problem in that some features might not contribute (possibly even harm) to the classification accuracy. Moreover, in our case, trimming the number of features is crucial; however, this should be done without diminishing classification accuracy. The reason for this constraint is that, as shown above, the vocabulary size may exceed billions of features, far more than the processing capacity of any abundant feature selection tool. Moreover, it is important to identify those terms that appear most frequently in files in order to avoid vectors that contain many zeros.” (Ex. 1004, p. 1487).

252. Menahem uses a similar approach to Tahan and explicitly defined ranges for keeping or rejecting features. Menahem first discloses a “TF-IDF measure” used for evaluating features:

Next, each n-gram term was represented using its Term Frequency (TF) and its Inverse Document Frequency (TF-IDF). The TF measure is the number of its appearances in the file, divided by the term with the maximal appearances. IDF is the log value of the number of files in the whole repository, divided by the number of files that includes the term. The TF-IDF measure is obtained by multiplying TF by IDF.
(Ex. 1004, p. 1487).

253. Menahem then teaches the use of that measure for selecting features to be kept or rejected:

Thus, we first extracted the top features based on the Document Frequency (DF) measure, which is the number of files the term was found in. We selected the top 0–5500 features which appear in most of the files, (those with highest DF scores) and the top 1000–6500 features. Finally, we used three methods of feature selection: Document Frequency, Fisher Score (Golub et al., 1999), and Gain Ratio (Mitchell, 1997) in order to choose the top 50, top 100, top 200 and top 300 n-grams, out of the 5500 n-grams, to be used for the evaluation.

After performing rigorous experiments in which we evaluated several inducers on the 192 different n-gram datasets, we chose the top 300 5-grams with TF-IDF values, top 300 6-grams with TF-IDF and top 300 6-grams, with TF datasets providing the best results.
(Ex. 1004, p. 1487).

254. A POSITA would have understood Menahem’s disclosure of “the top 300 5-grams with TF-IDF values” to represent ordering the “5-grams” based on their “TF-IDF values” and selecting items from the ordering using a predetermined threshold: “the top 300”. A POSITA would have understood from Menahem’s teachings that “TF-IDF values” represent values derived from n-grams (“static data points”) and would further have understood such values to teach the claimed “a value of one or more static data points from the plurality of extracted static data points”. Moreover, a POSITA would have recognized that Menahem’s feature selection process, which uses the “TF-IDF values” to select features based on defined thresholds, teaches the claimed “within a predetermined range”.

255. Menahem explains that the PE header features could be limited/reduced using a feature selection approach, Gain Ratio, similar to an approach disclosed by Tahan: “we applied the ***Gain Ratio feature selection method*** (Mitchell, 1997) on the n-gram-based and PE datasets.” (Ex. 1004 at p. 1488, emphasis added). Thus, Menahem also discloses this element.

256. A POSITA would have understood that Tahan or Menahem’s above-discussed techniques (e.g., feature selection/reduction techniques) that disclose this claim element are not limited to the specific examples (including specific features / static data points) discussed above. A POSITA would have understood that such techniques work with other features / static data points (e.g., those discussed in limitation 1[b]). For example, as discussed above, Menahem explains that feature selection/reduction techniques work with features / static data points such as n-grams and PE header fields. (See Ex. 1004 at p. 1488). Moreover, as discussed above, Tahan explains that the feature selection/reduction techniques are performed on a feature vector containing extracted features, which include (*inter alia*), features / static data points such as string features, PE header features, n-gram features, and Opcode n-gram features.

The techniques use three distinct stages:

1. ***Feature Extraction*** for file representation—The result of the feature extraction phase is a ***vector containing the features extracted***. An executable content is reduced or transformed into a more manageable form such as: (a) ***Strings . . .*** (b) ***Portable Executable File Format Fields . . .*** (c) ***n-gram . . .*** (d) ***Opcode n-gram . . .***

2. ***Feature Selection (or feature reduction)***—During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded.
(Ex. 1003, pp. 950-951, emphasis added).

257. Thus, a POSITA would have understood that the feature selection/reduction techniques would and should work with different types of features / static data points that are extracted and handled by the method/system. (See *id.*).

258. Also, to the extent Patent Owner incorrectly argues that such techniques do not work with different types of features / static data points, a POSITA would have been motivated to modify or add to such techniques such that they work with other features and static data points. For example, as discussed above, Tahan and Menahem explain the advantages of performing feature selection/reduction techniques (that disclose this claim element) to enhance the performance and accuracy of the method/system, for example, by ensuring that less relevant features / static data points have less impact on the machine learning classification process. For example, Tahan and Menahem explain:

2. Feature Selection (or feature reduction)—During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded. *Feature selection has many benefits including: improving the performance of learning modules by reducing the number of computations and as a result the learning speed; enhancing generalization capability; improving the interpretability of a model, etc.* (Ex. 1003, p. 951, emphasis added).

In ML applications, *the large number of features in many domains presents a huge problem in that some features might not contribute (possibly even harm) to the classification accuracy.* Moreover, in our case, trimming the number of features is crucial; however, this should be done without diminishing classification accuracy. The reason for this constraint is that, as shown above, the vocabulary size may exceed billions of features, far more than the processing capacity of any abundant feature selection tool. Moreover, it is important to identify those terms that appear most frequently in files in order to avoid vectors that contain many zeros. (Ex. 1004, p. 1487, emphasis added).

259. Thus, given the importance and effectiveness of the above-discussed feature selection/reduction techniques, a POSITA would have been motivated to configure such techniques to work with different features / static data points that are handled by the method/system. Moreover, as I discussed above, feature selection/reduction techniques were a common practice and would have been known to a POSITA. Thus, a POSITA would have been

capable of making such modifications (to the extent such modifications are needed) and would have had a reasonable expectation of success in making such modifications.

260. Although Tahan discloses this element, a POSITA would have also looked to Menahem for additional details. I have already explained in section IX.A. why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference.

261. Thus, Tahan alone or in combination with Menahem discloses this element.

262. Thus, Tahan in view of Menahem discloses limitation 1[c].

4. Limitation 1[d]

263. Limitation 1[d] of the '844 Patent reads:

evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

264. Tahan in view of Menahem discloses limitation 1[d] of the '844 Patent.

265. Tahan discloses that the use of support vector machines (which discloses the claimed “support vector processing”) to evaluate a feature vector and identify harmful executable files (malware) (“determine whether the executable file is harmful”) was common in the art. “In addition, in recent years many researchers have been using machine learning (ML) techniques to produce a **binary classifier that is able to distinguish malware from benign files**... The last phase is creating a classifier **using the reduced features vector created in phase 2 and a classification technique**. Among the many classification techniques, most of which have been implemented in the Weka platform (Witten and Frank, 2005), the following have been used in the context of **benign/malware files classification... support vector machines (SVMs)**...” (Ex. 1003, pp. 950-951, emphasis added).

266. As I discussed above (e.g., in Sections V.A. and VIII.A.) and as disclosed by the above quote, the last phase in machine learning processes (e.g., those disclosed by Tahan) is a classification step where an executable file is classified as benign or harmful, and thus whether an executable file is benign or harmful is determined. (See *id.*; see also Ex. 1003, pp. 950-951). Tahan explains (in the above quote) that an input into this last step is a feature vector from phase 2 (that I discussed above for limitation 1[c]). Thus, the classification step evaluates the feature vector discussed above. (See *id.*) Moreover, as I discussed above (e.g., in Sections V.A. and VIII.A.), extracted features are used in the classification step, and such extracted features are contained in a feature vector. (See *id.*) Thus, a POSITA would have understood that a feature vector is evaluated in the classification step (e.g., as discussed in more detail below, using support vector machine) to determine whether an executable file is harmful. (See *id.*; see also my analysis below).

267. Also, based on the disclosures (in the above quote and those below) that support vector machine is used (e.g., to determine if an executable file is harmful), a POSITA would have understood that a feature vector is evaluated using support vector machine (“support vector processing”), because as I discussed above for Section V.C., feature vectors generally are input into support vector machine for analysis/evaluation. (See *id.*; see also Section V.C.).

268. Moreover, the above quote states “. . . using machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files... [t]he last phase is creating a classifier using the reduced features vector created in phase 2 and a classification technique.” (Ex. 1003, pp. 950-951). Even if this disclosure is interpreted to mean that a feature vector is used/evaluated to create a classifier, which is used to determine if an executable file is harmful, Tahan still discloses this claim limitation. For example, as I discussed above (e.g., in

Sections V.A. and VIII.A.), creating or training a classifier involves analysis and evaluation of features (e.g., in a feature vector), and such an analysis/evaluation involves use of a machine learning algorithm. (See *id.*; Sections V.A. and VIII.A.). As disclosed by the above quote and as I explained below, a POSITA would have understood that support vector machine (“support vector processing”) can be used (and would have been motivated to use it) as such a machine learning algorithm. (See *id.*; see also my analysis below). Thus, Tahan discloses “evaluating the feature vector using support vector processing,” under this interpretation as well. (See *id.*). Moreover, such a created classifier is used to determine whether an executable file is harmful. (See *id.*). Thus, Tahan discloses “evaluating the feature vector using support vector processing to determine whether the executable file is harmful,” under this interpretation as well. (See *id.*).

269. As I explained in my overview of the reference, Menahem uses a mix of different classifiers to create a composite “ensemble” classifier. (See Section VIII.B.). Ensemble classification uses multiple component/base classifiers, potentially built using different techniques, or potentially with different data. When classifying an input, the data is processed in some way by the component/base classifiers and a final decision, or answer, is rendered by the aggregate classifier. For example, one approach is voting, wherein each component classifier gets a vote and the answer with the highest number of votes is selected as the answer for the overall classifier. Menahem teaches the use of SVMs, including as component/base classifiers in an ensemble classifier. “Ensemble of classifiers was evaluated on intrusion detection tasks as well. Mukkamalaa et al. (2005) used the Majority Voting schema to ensemble the results of multiple classifiers trained to detected intrusions within network traffic. The performance of the ensemble results was compared to the performance of the individual classifiers: SVM, MARS and three types of ANNs (RP, SCG and OSS)...” (Ex. 1004, p. 1484). Menahem further discloses that its

ensemble classifier technique builds on prior research about ensemble classifiers that used component/base classifiers (inducers) known in the art to be useful for malware detection, including Support Vector Machines (SVM). (See Ex. 1004 at p. 1484: “the datasets [of Kolter and Maloof (2006)] were evaluated using several inducers: IBK, Naïve Bayes, Support Vector Machines (SVM), Decision Trees, Boosted Naïve Bayes, Boosted SVMs, and Boosted Decision Tree algorithms”, “Mukkamalaa et al. (2005) used the Majority Voting schema to ensemble the results of multiple classifiers trained to detect intrusions within network traffic. The performance of the ensemble results was compared to the performance of the individual classifiers: SVM, MARS and three types of ANNs (RP, SCG and OSS)”); see also my analysis in Section VIII.B.).

270. I have already explained in section IX.A. why a POSITA would be motivated to combine Tahan and Menahem (e.g., including why a POSITA would be motivated to use support vector machines to determine if an executable file is harmful or not) and I incorporate this analysis by reference. As I explained in that section, a POSITA would have been motivated to combine Tahan’s Mal-ID classifier (and other classifiers including a classifier relating to support vector machines) with other classifiers into an ensemble as taught by Menahem. As Menahem explains, “[w]hen generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently... The outcome of this observation is that one should use multiple different classifiers... combining all the results, using an intelligent combination method into one final result.” (Ex. 1004 at p. 1484).

271. Tahan already teaches (and provides motivation for) combining Mal-ID with other classifiers and finds that the results were better in certain circumstances. Tahan explains that “[w]e attempted to improve the Mal-ID basic method by using Mal-ID features with various classifiers...” (Ex. 1003 at p. 959). Specifically, Tahan combines Mal-ID with three other

classification technologies known as “Rotation Forest,” “C4.5”, and a “multilayer perception.” (Ex. 1003 at p. 961). Tahan further identifies that other classifiers were known in the art to be used for classifying malware, including Support Vector Machines (SVM). (See Ex. 1003 at pp. 951-953: “the following have been used in the context of benign/malware files classification: ... support vector machines (SVMs) (Joachims, 1999)”, “Ye et al. (2009) [taught] ... a classification model is constructed from SVM ensemble with bagging”, “Dai et al. (2009) [found] ... In the evaluation the SVM model performed slightly better than the C4.5 model”; see also my analysis in Section VIII.A.).

272. In evaluating these classifier combinations, Tahan noted that no one approach was superior in all cases. “For each [classifier] method we have pinpointed its strong points and suggested cases where it should be preferred over the others.” (Ex. 1003 at p. 976). As I discussed above, use of support vector machines to detect malware such as harmful executable files (and its effectiveness) was well-known in the art. (See my analyses above and below for this limitation; see also Sections V.A, V.C, VIII.A-B, IX.A.). Thus, a POSITA would have been motivated to combine Mal-ID with other classifiers including a classifier for support vector machines.

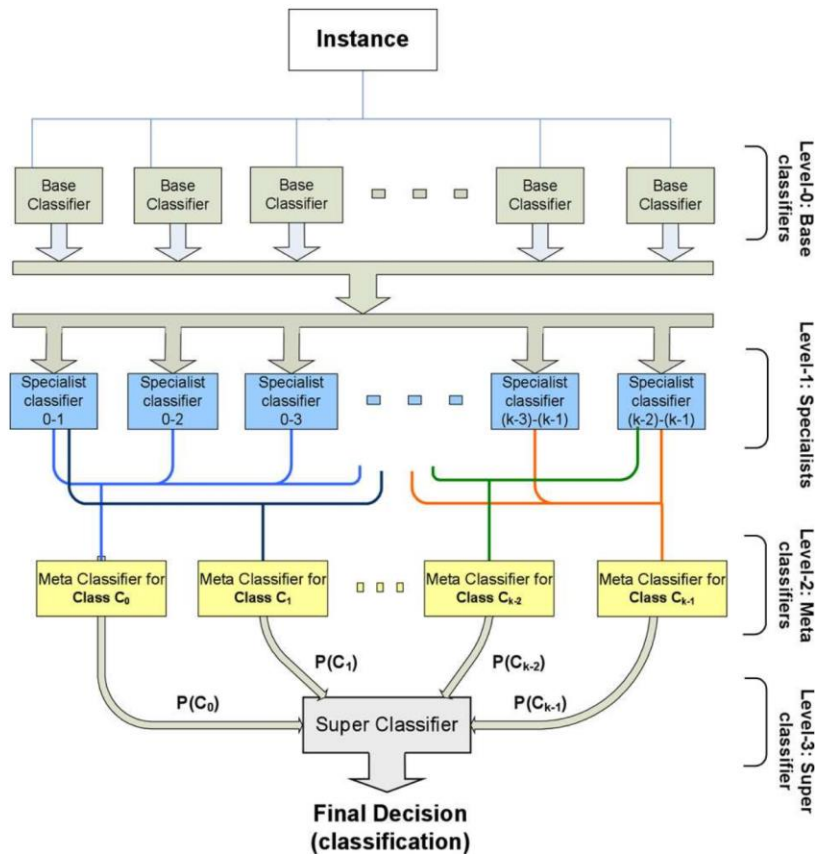
273. A POSITA would have been motivated to build on Tahan’s combination of Mal-ID with classifiers using Menahem’s more advanced ensemble combinations (including a classifier for support vector machines). (See *id.*).

274. To the extent that neither Tahan nor Menahem explicitly disclose using support vector processing (which is not conceded), it would have been obvious to a POSITA to do so for the reasons discussed above and in other sections. (See *id.*). Both Tahan and Menahem discuss the wide use of support vector machines (SVM) in the research art. (See *id.*).

275. For example, Tahan explains: “[t]he last phase is creating a classifier using the reduced features vector created in phase 2 and a classification technique. Among the many classification techniques... [is] support vector machines (SVM).” (Ex. 1003, p. 951). Tahan reviews a wide range of machine learning research that apply SVM classifiers to identifying malware. For example, Tahan identifies “Ye et al. (2009)”, “Dai et al. (2009)”, “Kolter and Maloof (2004)”, “Kolter and Maloof (2006)”, and “Henchiri and Japkowicz (2006)” are all examples of research into SVM classifiers and malware detection. (Ex. 1003 at pp. 951-953).

276. Menahem also refers to the use of SVMs, including in ensemble classifiers. “Ensemble of classifiers [sic] was evaluated on intrusion detection tasks as well. Mukkamalaa et al. (2005) used the Majority Voting schema to ensemble the results of multiple classifiers trained to detected intrusions within network traffic. The performance of the ensemble results was compared to the performance of the individual classifiers: SVM, MARS and three types of ANNs (RP, SCG and OSS)...” (Ex. 1004 at p. 1484).

277. Thus, a POSITA would have known that SVM’s can be used as base classifiers in ensemble classifiers such as Menahem’s. Menahem’s ensemble classifier architecture identifies “base classifier” generically as shown in the figure reproduced hereinbelow:



(Ex. 1004, p. 1486).

278. Also, to a POSITA, substituting SVM classifiers for another type of classifier would have been a known combination/technique with a predictable result (e.g., because as discussed above, the use of SVM classifiers to detect harmful executable files such as malware was well-known in the art).

279. A POSITA would have been motivated to use SVM classifiers. As noted by Menahem, “Future work may include the involvement of more inducers for base-classifiers (we had only used 5, while there are plenty more to choose from).” (Ex. 1004 at p. 1494). A POSITA would have known that SVM classifiers have many advantages. As I explained in my background on SVM, the SVM technique enables working with high-dimensions (large numbers of features) with low computational complexity. The dimensionality problem is referenced by both Tahan and

Menahem. Tahan explained, “Long n-gram create huge computational loads due to the number of features. This is known as the curse of dimensionality.” (Ex. 1003 at p. 952). Similarly, Menahem states, “in our case, trimming the number of features is crucial... [t]he reason for this constraint is that, as shown above, the vocabulary size may exceed billions of features, far more than the processing capacity of any abundant feature selection tool...” (Ex. 1004 at p. 1487).

280. Tahan also noted that SVM outperformed the C4.5 model. “In the evaluation the SVM model performed slightly better than the C4.5 model.” (Ex. 1003 at p. 952). Notably, both Tahan and Menahem used the C4.5 model. (See, e.g., Ex. 1003 at p. 960 and Ex. 1004 at p. 1487). A POSITA would have been motivated to try SVM classifiers (a well-known classifier that is effective) as alternative base classifiers in Menahem’s ensemble in order to take advantage of its ability to process large numbers of features and for its performance (which is known to be better at least than one of the other base classifiers).

281. Thus, Tahan in view of Menahem discloses limitation 1[d].

282. Therefore, I conclude that Tahan in view of Menahem renders Claim 1 obvious.

C. Claim 2 is rendered obvious by Tahan in view of Menahem

283. Claim 2 of the ’844 Patent is rendered obvious by Tahan in view of Menahem. This claim, which depends from Claim 1, reads:

The computer-implemented method according to claim 1, wherein the extracting the plurality of static data points further comprises classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classification.

284. As explained above with reference to Claim 1 and Sections VIII.A., VIII.B., and IX.A., Tahan and Menahem disclose extracting static data points of different types (e.g., strings,

byte sequences, numeric, nominal, Boolean, etc.). I incorporate by reference my analyses from Claim 1 and those sections.

285. For example, a component of Tahan's disclosures are binary strings extracted for analysis. Tahan discloses extracting these strings, known as n-grams. Tahan teaches that the n-grams, which are binary strings, can also represent character strings, such as specific lines of code. "In many cases 3- to 8-byte n-gram cannot represent even one line of code composed with a high level language. In fact, we showed in a previous paper (Tahan et al., 2010) that an n-gram should be at least 64 bytes long to uniquely identify a malware... Note that, when a segment represents at ***least one line of code developed using a high level language***; it can address the second deficiency of using short features that may be meaningless when considered alone." (Ex. 1003, p. 952). Thus, Tahan taught that, even though the extracted features (or static data points) are just n-gram sequences from within the file, the sequences represent data such as lines of code and other strings.

286. In another example, Tahan discloses a particular kind of n-gram (i.e., a binary string) that was used in the art. Tahan describes it as an "Opcode n-gram" wherein an "Opcode is a CPU specific operational code that performs specific machine instruction. Opcode n-gram refers to the concatenation of Opcodes into segments." (Ex. 1003, p. 951). A POSITA would have understood that Opcode n-grams are a binary character sequence, like the n-grams discussed in the preceding paragraph hereinabove, and that Opcode n-grams represent the same concept as what the '844 patent refers to as "program information [such as] APIs..." ('844 Patent at 5:67-6:1).

287. With respect to static data points that are numeric values, Tahan incorporates similar metrics related to file size that are used to identify "relative locations" in a file. "3-gram-relative-position-within-file: 224 entries, each of 64 bits. A bit value of 1 in a cell (i, j) indicates the appearance of 3-gram i in the jth internal segment of a file (assuming the file is divided into 64

equal length segments).” (Ex. 1003, p. 957). These are used in deriving an additional value about the file called the “spread”. As Tahan explains, “[e]ach segment comprises a number of 3-grams... spread approximates the maximum number of occurrences of a segment within different relative locations....” (Ex. 1003, p. 957).

288. Tahan also discloses a numeric value: entropy. Tahan defines entropy as: “the entropy measure of the bytes within a specific segment candidate... the entropy feature is also used to enable to identification of compressed areas (such as embedded JPEG images) and long repeating sequences that contain relatively little information.” (Ex. 1003 at p. 957). A POSITA would have understood this “measure” of the bytes to be a value associated with character sequences.

289. Tahan further discloses other examples of static data points of various types. Specifically, Tahan discloses that it was very common in the art to use data from executable file headers, such as PE file headers, as features, which are static data points:

Portable Executable File Format Fields—information embedded in Win32 and Win64-bit executables. The information is necessary for the Windows OS loader and application itself. Features extracted from PE executables may include all or part of the following pieces of information: attribute certificate—similar to checksum but more difficult to forge; date/time stamp; file pointer—a position within the file as stored on disk; linker information; CPU type; Portable Executable (PE) logical structure (including section alignment, code size, debug flags); characteristics—flags that indicate attributes of the image file; DLL import section—list of DLLs and functions the executable uses; export section—which functions can be imported by other applications; resource directory—indexed by a multiple-level binary-sorted tree structure (resources may include all kinds of information. For example, strings for dialogs, images, dialog structures; version information, build information, original filename, etc.); relocation table; and many other features.
(Ex. 1003, p. 950)

290. Tahan indicates that the use of PE data was common within the research community. In particular, Tahan identified research from “Schultz et al. (2001)”, “Elovici et al.

(2007)”, “Ye et al. (2008)”, “Ye et al. (2009)”, and “Ye et al. (2010)” as all using PE data in some form or another as features. (Ex. 1003, p. 953).

291. Tahan also discloses encoding the extracted static data points based on the type of data. For example, Tahan discloses a 3-gram feature encoded as just the raw bytes. “In particular, Ben-Gurion University Computational Center provided us malware that were detected by them over time. Each and every file from the collection is first broken into 3-grams (three consecutive bytes) and then an appropriate repository is constructed from the 3-grams.” (Ex. 1003 at 955).

292. Tahan further identified meta-features that are features of features. These meta-features represent values associated with the 3-grams. A POSITA would have understood meta-features to also be static data points, as they are values associated with the underlying strings. One of the meta features Tahan describes is “spread.” Spread is a numeric value derived from 3-grams. “[F]or each 3-gram, first retrieve the 3-gram-relative-position-within-file bit-field, and then perform ‘And’ operations over all the bit-fields and count the resulting number of bits that are equal to 1. In other words, spread approximates the maximum number of occurrences of a segment within different relative locations in train sets.” (Ex. 1003, p. 957). The “3-gram-relative-position-within-file bit-field” is also a static data point. It is encoded as a location within a file. “A bit value of 1 in a cell (i, j) indicates the appearance of 3-gram i in the jth internal segment of a file (assuming the file is divided into 64 equal length segments).” (Ex. 1003, p. 957).

293. As also explained above with reference to Claim 1 and Sections VIII.B. and IX.A, Menahem expands on Tahan with additional feature (static data point) types and encodings. It discloses extracting different component features: PE, function-based data, and binary sequences. The PE data includes (for example) nominal values and Boolean values, the function-based data includes (for example) numeric values, and the binary n-gram sequences are byte values. “We

developed a tool that extracts n-grams from the binary representation of a file. The tool parsed the files using a sliding window producing n-grams of varying lengths (denoted by n)... we [also] statically extracted different Portable Executable (PE) format features that represented information contained within each Win32 PE binary (EXE or DLL)... and we applied the Gain Ratio feature selection method (Mitchell, 1997) on the n-gram-based and PE datasets.” (Ex. 1004, pp. 1487-1488).

294. Each of these types of data (features / static data points) is encoded differently in that it is encoded into its own sub-component feature vector for classifiers. (See Sections VIII.B. and IX.A. and analysis for Claim 4 below). As I explained previously, Menahem creates a “super classifier” created from component classifiers. Menahem discloses that the “combined” feature vector for the super classifier could be composed of a combination of different feature vectors, each for its own classifier. “In future experiments we would like to evaluate the performance of inducers that are trained on different features.” (Ex. 1004, p. 1494). Menahem identifies that this was desirable. “When generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently” and contribute to improved performance of a super classifier that is an ensemble of classifiers. (Ex. 1004, p. 1484; see also *id.* at p. 1494; see also my analyses and explanations in Section VIII.B.).

295. Menahem explains, “In order to benefit from multiple different classifiers, and exploit their strengths we suggest using an ensemble method that will combine the results of the individual classifiers into one final result to achieve overall higher detection accuracy... The main goal is to find the best combining method...” (Ex. 1004, Abstract). In addition to combining different types of classifiers, Menahem also uses feature vectors comprised of different types of features, each encoded in an appropriate way. “We collected a large set of 22,735 benign files and

7690 malware that were identified as Win32 executable and DLLs. For each file **three types of features** were extracted: n-grams, Portable Executable (PE) features and Function-based features as described bellow. [sic]” (Ex. 1004, p. 1487, emphasis added).

296. Binary n-grams in the Menahem article are extracted directly from the binary representation of a file using a “sliding window” algorithm for each of the different values of n (3, 4, 5, and 6). These features are encoded based on the frequency of the n-gram. (Ex. 1004, p. 1487).

297. With reference to the Portable Executable (PE) data, Menahem teaches extracting and encoding data about the presence of data (a Boolean value), counts of data, and other references to contents/structure:

We statically extracted different Portable Executable (PE) format features that represented information contained within each Win32 PE binary (EXE or DLL)... The information extracted by this tool is as follows (total of 88 attributes):

- Information extracted from the PE-header that describes the details about physical structure of a PE binary (e.g., creation\modification time, machine type and file size)
- Optional PE-header information describing the logical structure of a PE binary (e.g., linker version, section alignment, code size and debug flags)
- Import Section: which DLLs were imported and which functions from the imported DLLs that were used
- Export Section: which functions were exported (if the file being examined is a DLL)
- Resource Directory: resources used by a given file (e.g., dialogs and cursors)
- Version Information: (e.g., internal and external name of a file, version number)

(Ex. 1004, pp. 1487-1488).

298. With respect to “function-based features,” Menahem explains that “we use a j48 decision tree classifier in order to mark the beginning and ending of functions residing in the binary

representation of the files.” (Ex. 1004 at p. 1488). Using these function marks, Menahem extracts the following attributes/features, which are derived from an executable file:

- Size of file
- File’s entropy value
- Total number of detected functions
- Size of the longest function that was detected
- Size of the shortest function that was detected
- Average size of detected functions
- Standard deviation of the size of detected functions
- Number of functions divided into fuzzy groups by its length in bytes (16, 24, 40, 64, 90, 128 and 256 bytes)
- Function ratio — the proportion of number of detected functions from the overall size of the file
- Code ratio — the proportion of the size (in bytes) of detected functions from the overall size of the file.

(Ex. 1004, p. 1488).

299. Thus, as discussed above (and for Claims 1 and 4 and Sections VIII.A., VIII.B., and IX.A.), the Tahan-Menahem method/system (which uses the above-discussed ensemble of classifiers approach) uses different types of classifiers (e.g., classifier algorithms), which use different feature vectors containing different types of features (static data points), each encoded in an appropriate way for later analysis by classifiers (and/or by the ensemble of classifiers also referred to as the super classifier). In order to do that, as discussed above, the Tahan-Menahem method/system identifies and recognizes the different types of features (static data points), and thus “classif[ies] the extracted static data points according to a type of data extracted from the executable file.” And, as discussed above, such different types of features (static data points) are encoded for classifiers (e.g., classifier algorithms) based on the data types (e.g., classification of data types). Thus, the Tahan-Menahem method/system discloses “classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classification.” To the extent Patent Owner incorrectly argues that the Tahan-Menahem method/system does not disclose this

technique, a POSITA would have been motivated to implement such a technique because the Tahan-Menahem method/system uses different classifiers (as ensemble of classifiers) that handle different types of features (static data points). Thus, a POSITA would have understood that it was necessary or desirable to identify and recognize the different types of extracted features by classifying them so that they can be processed (e.g., encoded) accordingly and appropriately for the different classifiers.

300. Moreover, a POSITA would have understood that the Tahan-Menahem method/system performs the above-discussed “classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classification” as part of the feature (static data point) extraction process, because the extracted features need to be and are encoded accordingly to be included in feature vectors for the classifiers. Or, a POSITA would have been motivated to implement the Tahan-Menahem method/system in such a manner so that the extracted static data points can be classified and processed/encoded along with (or following) the extraction process, so that they are prepared appropriately before the step of generating feature vectors from extracted static data points. Thus, a POSITA would have understood that the Tahan-Menahem method/system discloses “the extracting the plurality of static data points further comprises classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classification,” or would have been motivated to configure the Tahan-Menahem method/system in such a manner.

301. I have already explained in section IX.A. and Claim 1 why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference. For this particular claim, a POSITA would have recognized that by categorizing inputs and generating

encodings for component classifiers in the ensemble, a wider range of classifiers can be used as components. Thus, Menahem enables Tahan to be broader and more flexible in combining different types of classifiers together, which would have improved Tahan and the Tahan-Menahem method/system.

302. Thus, I conclude that Tahan in view of Menahem renders obvious Claim 2.

D. Claim 3 is rendered obvious by Tahan in view of Menahem

303. Claim 3 of the '844 Patent is rendered obvious by Tahan in view of Menahem. This claim, which depends from Claim 2, reads:

The computer-implemented method according to claim 2, wherein generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.

304. As explained with respect to limitation 1[c], Tahan alone or in combination with Menahem discloses selectively turning on or off features ***of a feature vector*** based on values of extracted static data points. I incorporate by reference my analyses for limitation 1[c]. Tahan in view of Menahem further discloses “selectively setting features ***of the classifier*** based on the plurality of static data points.”

305. Tahan teaches turning on or off features of a classifier and describes the feature selection/reduction process, which a POSITA would have understood includes selectively setting (e.g., turning on or off) features, as a common technique when using machine learning to classify a file and to detect malware.

In addition, in recent years many ***researchers have been using machine learning (ML) techniques to produce a binary classifier that is able to distinguish malware from benign files.***

The techniques use three distinct stages:

...

2. Feature Selection (or feature reduction)—During this phase the vector created in phase 1 is evaluated and ***redundant and irrelevant features are discarded***. **Feature selection has many benefits including: improving the performance of learning modules by reducing the number of computations and as a result the learning speed;** enhancing generalization capability; improving the interpretability of a model, etc. Feature selection can be done using a wrapper approach or a correlation-based filter approach (Mitchell, 1997). Typically, the filter approach is faster than the wrapper approach and is used when many features exist. ***The filter approach uses a measure to quantify the correlation of each feature, or a combination of features, to a class.*** The overall expected contribution to the classification is calculated and selection is done according to the highest value. ***The feature selection measure can be calculated using many techniques, such as gain ratio (GR); information-gain (IG); Fisher score ranking technique (Golub et al., 1999) and hierarchical feature selection (Henchiri and Japkowicz, 2006).*** (Ex. 1003, pp. 950-951, emphasis added).

306. Thus, Tahan makes it clear that selectively setting (e.g., turning on or off) features was a common practice and would have been known to a POSITA.

307. Moreover, Tahan discloses turning on or off features “based on the plurality of static data points.” For example, Tahan discloses using feature sorting to turn on and off features based on values of features (static data points). In this scenario, Tahan discloses computing a ranking or score for each feature (static data point) based on its value (e.g., to determine “[t]he overall expected contribution to the classification” process (Ex. 1003, p. 951)), and only features within a predetermined ranking are enabled (e.g., turned on to be used in subsequent processes such as evaluating a feature vector to determine if a file is harmful). Thus, features are selectively turned on or off if they are within (or outside) a predetermined range (and those features that are turned off will have less impact on the later classification process).

308. Tahan describes this process this way:

The filter approach uses a measure to quantify the correlation of each feature, or a combination of features, to a class. The overall expected contribution to the classification is calculated and selection is done according to the highest value. The feature selection measure can be

calculated using many techniques, such as gain ratio (GR); information-gain (IG); Fisher score ranking technique (Golub et al., 1999) and hierarchical feature selection (Henchiri and Japkowicz, 2006). (Ex. 1003, p. 951, emphasis added).

309. Tahan discloses using the “Gain Ratio feature selection” process. “We will refer to our variation of Kolter and Maloof method as GR500BDT as it uses ***Gain Ratio feature selection***, 500 4-grams, and Boosted Decision Tree classifier.” (Ex. 1003, p. 961, emphasis added).

310. Menahem also discloses this limitation. Menahem, for example, provides motivation for narrowing the features used by the classifier: “In ML applications, the large number of features in many domains presents a huge problem in that some features might not contribute (possibly even harm) to the classification accuracy. Moreover, in our case, trimming the number of features is crucial; however, this should be done without diminishing classification accuracy. The reason for this constraint is that, as shown above, the vocabulary size may exceed billions of features, far more than the processing capacity of any abundant feature selection tool. Moreover, it is important to identify those terms that appear most frequently in files in order to avoid vectors that contain many zeros.” (Ex. 1004, p. 1487).

311. Menahem uses a similar approach to Tahan and explicitly defined ranges for keeping or rejecting features in the classifier. Menahem first discloses a “TF-IDF measure” used for evaluating features:

Next, each n-gram term was represented using its Term Frequency (TF) and its Inverse Document Frequency (TF-IDF). The TF measure is the number of its appearances in the file, divided by the term with the maximal appearances. IDF is the log value of the number of files in the whole repository, divided by the number of files that includes the term. The TF-IDF measure is obtained by multiplying TF by IDF. (Ex. 1004, p. 1487).

312. Menahem then teaches the use of that measure for selecting which features are to be used in the classifier:

Thus, we first extracted the top features based on the Document Frequency (DF) measure, which is the number of files the term was found in. We selected the top 0–5500 features which appear in most of the files, (those with highest DF scores) and the top 1000–6500 features. Finally, we used three methods of feature selection: Document Frequency, Fisher Score (Golub et al., 1999), and Gain Ratio (Mitchell, 1997) in order to choose the top 50, top 100, top 200 and top 300 n-grams, out of the 5500 n-grams, to be used for the evaluation.

After performing rigorous experiments in which we evaluated several inducers on the 192 different n-gram datasets, we chose the top 300 5-grams with TF-IDF values, top 300 6-grams with TF-IDF and top 300 6-grams, with TF datasets providing the best results.
(Ex. 1004, p. 1487).

313. A POSITA would have understood Menahem’s disclosure of “the top 300 5-grams with TF-IDF values” to represent ordering the “5-grams” based on their “TF-IDF values” and selecting items from the ordering using a threshold: “the top 300”. A POSITA would have understood from Menahem’s teachings that “TF-IDF values” represent values derived from n-grams (“static data points”) and would further have understood that such values, because they are based on the n-grams, teach the claimed “based on the plurality of extracted static data points”. Moreover, a POSITA would have recognized that Menahem’s feature selection process, which uses the “TF-IDF values” to select features of a classifier based on defined thresholds, teaches the claimed “selectively setting features of the classifier”.

314. Menahem explains that the PE header features could be limited/reduced using a feature selection approach, Gain Ratio, similar to an approach disclosed by Tahan: “we applied the ***Gain Ratio feature selection method*** (Mitchell, 1997) on the n-gram-based and PE datasets.”
(Ex. 1004, p. 1488, emphasis added).

315. Thus and for the reasons discussed in limitation 1[c], a POSITA would have understood that Tahan and Menahem disclose selectively turning on or off features of a feature vector based on values of extracted static data points (through the above-discussed feature selection/reduction processes). Moreover, a POSITA would have understood that this also discloses this claim. For example, a POSITA would have recognized that selectively turning on or off features of a feature vector based on values of extracted static data points teaches similar, related, and compatible techniques for “selectively setting features of the classifier”. (See, e.g., ’844 Patent at 7:57-62 (“As an example, the learning classifier can adaptively set **features of a learning classifier** based on the static data points extracted for a file. For example, ranges of data can be identified for static data points that may **enable the learning classifier to controllably select (e.g., turn on/off) features** for evaluation by learning classifier.”), emphasis added).

316. Moreover, because the Tahan-Menahem method/system’s processes are performed by machine learning mechanisms/modules/applications (e.g., similar to the broader “classifier component”) that disclose the “classifier” recited by Claim 1 and this claim (see my analyses for Claim 1), the above-discussed feature selection/reduction process is also performed by the said classifier. Moreover, the Tahan-Menahem method/system’s processes (including handling/processing of features) are also performed by the said classifier. Thus, the features handled and processed by the classifier (including features that go through the above-discussed process of feature selection/reduction) can be understood to be “features of the classifier.” Thus, the above-discussed process of feature selection/reduction (selectively turning on or off features of a feature vector (and the classifier) based on values of extracted static data points) discloses “selectively setting features of the classifier based on the plurality of extracted static data points.”

317. A POSITA would have been motivated to configure the Tahan-Menahem method/system such that “selectively setting features of the classifier based on the plurality of extracted static data points” occurs along with (or following) the process of “generating the at least one feature vector” because the feature selection/reduction process concerns feature vectors (and features therein). Given that both processes are closely related to feature vectors, a POSITA would have been motivated to configure the Tahan-Menahem method/system such that “generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.”

318. I have already explained in section IX.A. and Claim 1 why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference.

319. Thus, I conclude that Tahan in view of Menahem renders obvious Claim 3.

E. Claim 4 is rendered obvious by Tahan in view of Menahem

320. Claim 4 of the '844 Patent is rendered obvious by Tahan in view of Menahem. This claim, which depends from Claim 2, reads:

The computer-implemented method according to claim 2, wherein the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.

321. As explained above with reference to Claim 1 and Sections VIII.A., VIII.B., and IX.A. and as I explain below, Tahan and Menahem disclose extracting static data points of different types (e.g., strings, byte sequences, numeric, nominal, Boolean, etc.). I incorporate by reference my analyses from Claim 1 and those sections. Also, as I explained above with reference to Claim 2, the Tahan-Menahem method/system discloses “classifying the extracted static data points according to a type of data extracted from the executable file,” such as string, byte sequence,

numeric, nominal, and Boolean values. I incorporate by reference my analyses from Claim 2. Thus, the Tahan-Menahem method/system discloses “the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.”

322. For example, a component of Tahan’s disclosures are binary strings extracted for analysis. Tahan discloses extracting these strings, known as n-grams. Tahan teaches that the n-grams, which are binary strings, can also represent character strings, such as specific lines of code. “In many cases 3- to 8-byte n-gram cannot represent even one line of code composed with a high level language. In fact, we showed in a previous paper (Tahan et al., 2010) that an n-gram should be at least 64 bytes long to uniquely identify a malware... Note that, when a segment represents at ***least one line of code developed using a high level language***; it can address the second deficiency of using short features that may be meaningless when considered alone.” (Ex. 1003, p. 952, emphasis added). Thus, Tahan taught that, even though the extracted features (or static data points) are just n-gram sequences from within the file, the sequences represent data such as lines of code and other strings.

323. In another example, Tahan discloses that an alternative form of binary n-grams was used in the art. Tahan describes these as “Opcode n-gram[s]” wherein an “Opcode is a CPU specific operational code that performs specific machine instruction. Opcode n-gram refers to the concatenation of Opcodes into segments.” (Ex. 1003, p. 951). A POSITA would have understood that Opcode n-grams are a binary character sequence, and that they represent the same concept as what the ’844 patent refers to as “program information [such as] APIs...” (’844 Patent at 5:67-6:1).

324. With respect to static data points which are numeric values, Tahan incorporates similar metrics related to file size that are used to identify “relative locations” in a file. “3-gram-relative-position-within-file: 224 entries, each of 64 bits. A bit value of 1 in a cell (i, j) indicates the appearance of 3-gram i in the jth internal segment of a file (assuming the file is divided into 64 equal length segments).” (Ex. 1003, p. 957). These are used in deriving an additional value about the file called the “spread”. As Tahan explains, “[e]ach segment comprises a number of 3-grams... spread approximates the maximum number of occurrences of a segment within different relative locations....” (Ex. 1003, p. 957).

325. Tahan also discloses a numeric value: entropy. Tahan defines entropy as: “the entropy measure of the bytes within a specific segment candidate... the entropy feature is also used to enable to identification of compressed areas (such as embedded JPEG images) and long repeating sequences that contain relatively little information.” (Ex. 1003, p. 957). A POSITA would have understood this “measure” of the bytes to be a value associated with character sequences.

326. Tahan further discloses other examples of static data points of various types. Specifically, Tahan discloses that it was very common in the art to use data from executable file headers, such as PE file headers, as features, which are static data points:

Portable Executable File Format Fields—information embedded in Win32 and Win64-bit executables. The information is necessary for the Windows OS loader and application itself. Features extracted from PE executables may include all or part of the following pieces of information: attribute certificate—similar to checksum but more difficult to forge; date/time stamp; file pointer—a position within the file as stored on disk; linker information; CPU type; Portable Executable (PE) logical structure (including section alignment, code size, debug flags); characteristics—flags that indicate attributes of the image file; DLL import section—list of DLLs and functions the executable uses; export section—which functions can be imported by other applications; resource directory—indexed by a multiple-level binary-sorted tree structure (resources may include all kinds of

information. For example, strings for dialogs, images, dialog structures; version information, build information, original filename, etc.); relocation table; and many other features.
(Ex. 1003, p. 950)

327. Tahan indicates that the use of PE data was common within the research community. In particular, Tahan identified research from “Schultz et al. (2001)”, “Elovici et al. (2007)”, “Ye et al. (2008)”, “Ye et al. (2009)”, and “Ye et al. (2010)” as all using PE data in some form or another as features. (Ex. 1003, p. 953).

328. Tahan also discloses encoding the extracted static data points based on the type of data. For example, Tahan discloses a 3-gram feature encoded as just the raw bytes. “In particular, Ben-Gurion University Computational Center provided us malware that were detected by them over time. Each and every file from the collection is first broken into 3-grams (three consecutive bytes) and then an appropriate repository is constructed from the 3-grams.” (Ex. 1003 at 955).

329. Tahan further identifies meta-features that are features of features. These meta features represent values associated with the 3-grams. A POSITA would have understood that meta-features to also be static data points, as they are values associated with the underlying strings. One of the meta features Tahan describes is “spread.” Spread is a numeric value derived from 3-grams. “for each 3-gram, first retrieve the 3-gram-relative-position-within-file bit-field, and then perform ‘And’ operations over all the bit-fields and count the resulting number of bits that are equal to 1. In other words, spread approximates the maximum number of occurrences of a segment within different relative locations in train sets.” (Ex. 1003 at p. 957). The “3-gram-relative-position-within-file bit-field” is also a static data point. It is encoded as a location within a file. “A bit value of 1 in a cell (i, j) indicates the appearance of 3-gram i in the jth internal segment of a file (assuming the file is divided into 64 equal length segments).” (Ex. 1003, p. 957).

330. As I also explained with respect to Claim 1, Menahem classifies extracted data points as numeric values, nominal values, strings, and Boolean values when extracting the data from PE headers. Menahem explains that these values are values and strings. “Certain parts of EXE files might indicate that a file is affected by a virus. This may be an inconsistency between different parts of **version numbers** or **internal/external name of a file.**” (Ex. 1004, p. 1487, emphasis added). Menahem identifies 88 attributes of PE headers extracted from PE headers and 17 attributes from function-based features. Examples of these features include numerical values such as “file size”, nominal values such as “linker version”, strings such as “machine type”, and Boolean values such as “debug flags.” (Ex. 1004, pp. 1487-1488). The PE components are encoded into one feature vector, and the function-based features are encoded into another. (Ex. 1004, Table 1).

331. On the other hand, Menahem classifies n-grams as binary data: “We developed a tool that extracts n-grams from the binary representation of a file.” (Ex. 1004, p. 1487). These binary features are included in a separate feature vector.

332. Thus, Menahem discloses categorizing/classifying PE features as nominal values like “linker version” and Boolean values such as “debug flags.” It categorizes function-based values as numeric values such as “total number of detected functions.” And, of course, binary n-grams are categorized as byte sequences. At the very least, this categorizes three of the four categories recited by the claim. In my opinion, it would have been obvious to a POSITA that nominal values and Boolean values could be broken up into their own separate vectors, and thus categories. A POSITA would have recognized that Menahem’s divisions are non-limiting and any number of divisions are possible.

333. Menahem also explicitly discloses a separate Boolean classifier: the meta-classifier output of component machine learning classifiers. Specifically, in the combined ensemble algorithm, component machine learning classifiers each output a result, which is a classification (e.g., “malicious” or “benign”). These outputs become the input to the super classifier or to the meta-inducer that creates the super classifier. As Menahem explains, “The idea is to create a meta-dataset containing a tuple for each tuple in the original dataset. However, instead of using the original input attributes, it uses the ***predicted classifications by the classifiers as the input attributes.***” (Ex. 1004, p. 1486, emphasis added). For binary classification (e.g., as “malicious” or “benign”), this is a Boolean output and the feature vector created from the output is a vector of Boolean values. A POSITA would have understood that these Boolean values were a meta-feature, similar to Tahan’s meta-features. Like Tahan, a POSITA would have recognized Menahem’s meta-features as values derived from or otherwise associated with the underlying character strings. Thus, a POSITA would have understood that the Tahan-Menahem method/system classifies features (static data points) into categories, including a Boolean category (and the others discussed above), or at least would have been motivated to configure the method/system in such a manner.

334. For reasons stated above, the Tahan-Menahem method/system discloses this limitation. To the extent Patent Owner incorrectly argues that the method/system does not disclose this limitation, a POSITA would have been motivated to implement the features of this limitation. For example, as discussed above in this section and for Claims 1 and 2 and Sections VIII.A., VIII.B., and IX.A., the Tahan-Menahem method/system (which uses the above-discussed ensemble of classifiers approach) uses different types of classifiers (e.g., classifier algorithms), which use different feature vectors containing different types of extracted features (static data

points), wherein such types include strings, byte sequences, numeric, nominal, and Boolean. Thus, a POSITA would have found it obvious that when the Tahan-Menahem method/system classifies different types of extracted features (static data points), it does so into categories such as strings, byte sequences, numeric, nominal, and Boolean.

335. I have already explained in section IX.A. and Claim 1 why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference. For this particular claim, a POSITA would have recognized that by categorizing inputs and generating encodings for component classifiers in the ensemble, a wider range of classifiers can be used as components. Thus, Menahem enables Tahan to be broader and more flexible in combining different types of classifiers together, which would have improved Tahan and the Tahan-Menahem method/system.

336. Thus, I conclude that Tahan in view of Menahem renders obvious Claim 4.

F. Claim 5 is rendered obvious by Tahan in view of Menahem

337. Claim 5 of the '844 Patent is rendered obvious by Tahan in view of Menahem. This claim, which depends from Claim 4, reads:

The computer-implemented method according to claim 4, wherein analyzing the executable file further comprises generating at least one feature vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.

338. This claim has unusual language in that it says “wherein analyzing the executable...” even though the word “analyzing” has not appeared previously in the claims. I have interpreted this to mean the entire processing of at least Claim 1, and perhaps also the preceding dependent Claim 2 and Claim 4. As I discussed above for Claims 1, 2, and 4, the processes of Tahan and Menahem that disclose the processes of those claims include “generating at least one

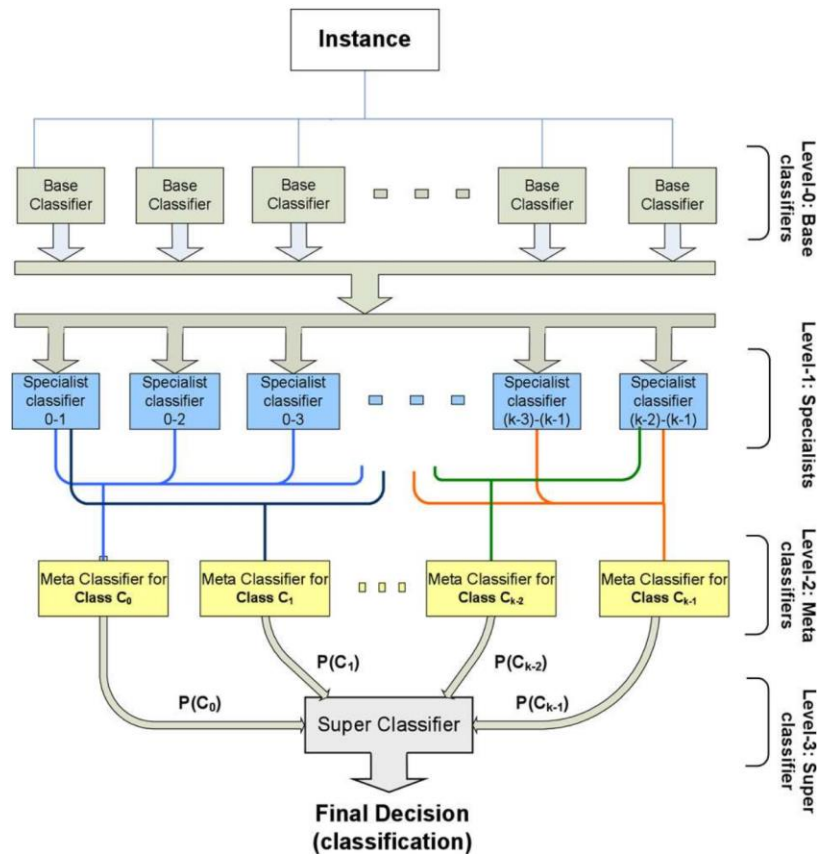
feature vector from the plurality of extracted static data points.” Thus, Tahan and Menahem disclose “wherein analyzing the executable file further comprises generating at least one feature vector from the plurality of extracted static data points.”

339. Moreover, Tahan and Menahem disclose “wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.” For example, as I explained with respect to Claims 1, 2, and 4, Tahan and Menahem disclose classifying different types of features and encoding them. For example, Tahan and Menahem disclose “classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values” (of Claim 4). I incorporate by reference my analyses for Claims 1, 2, and 4. Tahan also discloses that it was known in the art that different types of classifiers could be used for working with different encodings of features, such as Rotation Forest wherein “the features used in the various nodes are linear combination of the original features.” (Ex. 1003, p. 970).

340. As I also explained with respect to Claims 1, 2, and 4, Menahem discloses an “ensemble” classifier that takes inputs from component classifiers, such as the classifiers disclosed in Tahan. Menahem teaches that different features can be used as inputs to the ensemble classifier. “In future experiments we would like to evaluate the performance of inducers that are trained on different features.” (Ex. 1004, p. 1494). Menahem identifies that this was desirable. “When generating multiple classifiers using different inducers and various features (that are extracted from the same set of instances) each classifier will perform differently” and contribute to improved performance of a super classifier that is an ensemble of classifiers. (Ex. 1004, p. 1484; see also *id.* at p. 1494; see also my analyses and explanations in Section VIII.B.). When different feature (static data point) types (e.g., PE vs. n-grams vs. function-based features and/or numeric vs.

nominal vs. string or byte sequences vs. Boolean) are used within the ensemble, the different feature types are weighted. Specifically, the individual classifiers, each of which is using a different set of features, are weighted. In one method, the result of each classifier is given a vote, and the decision with the most votes is chosen as the output of the aggregate. (Ex. 1004, pp. 1484, 1485). Another approach is called “performance weighting.” In this approach, “the weight of each classifier is set proportional to its accuracy performance on a validation set.” (*Id.*). Other approaches such as “Conditional Probability” and “Stacking” also use various approaches to weight the output of one of the sub-classifiers. (Ex. 1004, pp. 1485-1486).

341. In Troika, the ensemble classifier disclosed by Menahem, the weighted classifiers which take different feature (static data point) types as input effectively weight based on the classified categories and the static data points. The architecture of Troika is represented hereinbelow:



(Ex. 1004, p. 1486).

342. As shown in the architecture above, classification is adjusted by probability (denoted $P(X)$) for different classes. Menahem explains that stacking, on which Troika was built, uses “a meta-learner... to induce which [base] classifiers are reliable and which are not.” (Ex. 1004, p. 1486).

343. Moreover, as discussed above for limitation 1[c], Tahan and Menahem disclose feature selection/reduction, where features of a feature vector are selectively turned on or off based on values of extracted static data points. I incorporate by reference my analyses for limitation 1[c]. There, I explained that the feature selection/reduction process results in some features having more impact on the classification step in which a classifier determines if an executable file is harmful, and some features having less impact. Thus, a POSITA would have understood (or at least would

have found obvious) that the feature selection/reduction process results in “features of the generated feature vector [being] weighted based at least upon . . . the plurality of extracted static data points.”

344. Therefore, Tahan and Menahem disclose or render obvious “generating at least one feature vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points” and this claim.

345. I have already explained in section IX.A. and Claim 1 why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference. For this particular claim, a POSITA would have recognized that by weighting inputs in the ensemble, more trustworthy classifiers can have a bigger impact on the final classification. Thus, Menahem enables Tahan to be broader and more flexible in combining different types of classifiers together, which would have improved Tahan and the Tahan-Menahem method/system.

346. Thus, I conclude that Tahan in view of Menahem render obvious Claim 5.

G. Claim 8 is rendered obvious by Tahan in view of Menahem

347. Claim 8 of the '844 Patent is rendered obvious by Tahan in view of Menahem. This claim, which depends from Claim 1, reads:

The computer-implemented method according to claim 1, wherein the plurality of static data points is extracted using a machine learning technique.

348. Tahan in view of Menahem renders obvious Claim 8. For example, as I discussed above for limitation 1[c], Tahan and Menahem disclose feature selection/reduction where the process selects/reduces features such that fewer, more relevant features are ultimately used in the classification step that determines if an executable file is harmful. In limitation 1[c], feature selection/reduction was discussed in the context of selecting/reducing features of a feature vector.

However, a POSITA would have understood that the concepts and techniques of feature selection/reduction (e.g., discussed in limitation 1[c]) can be applied to and/or implemented in a related context such as for extracting features from an executable file.

349. For example, in my opinion, a POSITA would have understood that the “Gain Ratio” machine learning technique/method (for feature selection/reduction) disclosed by Tahan and Menahem can also be applied to the process of extracting features so that fewer, more relevant features can be extracted. For example, Tahan disclosed, “We will refer to our variation of Kolter and Maloof method as GR500BDT as it uses Gain Ratio feature selection, 500 4-grams, and Boosted Decision Tree classifier.” (Ex. 1003, p. 961). Tahan explained the core need for a technique, such as Gain Ratio, to narrow down features to those that most impacted the classification.

Feature Selection (or feature reduction)—During this phase the vector created in phase 1 is evaluated and redundant and irrelevant features are discarded... Feature selection can be done using a wrapper approach or a correlation-based filter approach (Mitchell, 1997)... The filter approach uses a measure to quantify the correlation of each feature, or a combination of features, to a class. The overall expected contribution to the classification is calculated and selection is done according to the highest value. The feature selection measure can be calculated using many techniques, ***such as gain ratio (GR)***... (Ex. 1003, p. 951, emphasis added).

350. Menahem also discloses feature selection/reduction using the Gain Ratio machine learning technique. “Finally, we used three methods of feature selection: Document Frequency, Fisher Score (Golub et al., 1999), ***and Gain Ratio (Mitchell, 1997)*** in order to choose the top 50, top 100, top 200 and top 300 n-grams, out of the 5500 n-grams, to be used for the evaluation... After performing rigorous experiments in which we evaluated several inducers on the 192 different n-gram datasets, we chose the top 300 5-grams with TF–IDF values, top 300 6-grams with TF–IDF and top 300 6-grams, with TF datasets providing the best results.” (Ex. 1004, p. 1487).

351. A POSITA would have been motivated to apply the feature selection/reduction concepts and techniques (e.g., including those utilizing machine learning techniques/methods) to extraction of features from an executable file, because such a modified extraction process would have resulted in a fewer and more relevant features being used in the classification step (where whether an executable file is harmful is determined), and thus would have contributed to the efficiency and/or effectiveness of the method/system (e.g., similar to how the feature selection/reduction process discussed in limitation 1[c] would have improved the efficiency and/or effectiveness of the method/system). A difference between the feature selection/reduction process for the feature extraction process as opposed to for the process discussed in limitation 1[c] is that the former performs the feature selection/reduction for the extraction step, whereas the latter performs it later (e.g., on features of a feature vector). A POSITA would have understood that one may prefer one over the other based on different situations and setups. Thus, a POSITA would have been motivated to apply the feature selection/reduction concepts and techniques (e.g., including those utilizing machine learning techniques/methods) to the feature extraction step under certain circumstances and setups, or would have been motivated to try it to see if the modified method/system performs better under given circumstances and setups.

352. I have already explained in section IX.A. and Claim 1 why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference.

353. Thus, I conclude that Tahan in view of Menahem renders obvious Claim 8.

H. Claim 10 is rendered obvious by Tahan in view of Menahem

354. Claim 10 of the '844 Patent is nearly identical to Claim 1.

Claim 1	Claim 10
---------	----------

A computer-implemented method comprising:	A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:
extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file;	extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file;
generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on	generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on whether one or more values

whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range; and	of one or more static data points from the plurality of extracted static data points is within a predetermined range; and
evaluating the feature vector using support vector processing to determine whether the executable file is harmful.	evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

355. Thus, as shown above, the main difference between Claim 1 and Claim 10 is that Claim 10 recites “[a] computer-readable storage medium containing the instructions, that when executed on at least one processor, causing the processor to execute” the steps of the Claim (the preamble).

356. I understand that the preamble of a patent claim is not always considered limiting. However, to the extent that the preamble of Claim 10 of the ’844 Patent is considered limiting, Tahan in view of Menahem discloses the preamble of Claim 10.

357. Tahan discloses that its algorithm were implemented as instructions stored in a computer-readable storage medium and executed by a processor. Tahan explains, “The evaluation computer used an Intel Q6850 CPU with 4GB of RAM. The processing time was measured using only 1 CPU core, although the implemented algorithm natively supported multiple cores.” (Ex. 1003 at p. 962). A POSITA would have understood that running a program on a computer with a processor (such as an Intel Q6850 CPU) involves having the program instructions be stored on a computer readable storage medium and executed by the processor (to perform the algorithms).

358. Menahem discloses that its algorithms were “implemented in WEKA (Witten and Frank, 2005) in the JAVA programming language.” (Ex. 1004 at p. 1487). A POSITA would have understood that JAVA programs contain instructions, and in order to be executed, they are stored in a computer-readable storage medium. A POSITA would further understand that such instructions are executed by a processor (to perform the algorithms).

359. Although Tahan discloses this element, a POSITA would have also looked to Menahem for additional details. I have already explained in section IX.A. and Claim 1 why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference.

360. The motivation to combine Tahan and Menahem for other limitations of Claim 10 are identical to the motivation to combine them for Claim 1. I incorporate all other analysis related to Claim 1 into my analysis of Claim 10 by reference. Tahan in view of Menahem discloses each limitation in Claim 10 based on the same analysis I performed for each corresponding limitation in Claim 1.

361. Thus, I conclude that Tahan in view of Menahem renders Claim 10 obvious.

I. Claims 11-14 are rendered obvious by Tahan in view of Menahem

362. Claims 11 through 14 are effectively identical to Claims 2-5. In the table below, I visualize the minor differences between them. Changes in Claims 11 through 14 are bolded.

Claim 2. The computer-implemented method according to claim 1, wherein the extracting the plurality of static data points further comprises classifying the extracted static data points according to	Claim 11. The computer-readable storage device according to claim 10, wherein the extracting of the plurality of static data points executed by the processor further comprises classifying
---	--

a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classification.	the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classifying.
Claim 3. The computer-implemented method according to claim 2, wherein generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.	Claim 12. The computer-readable storage device according to claim 11, wherein generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.
Claim 4. The computer-implemented method according to claim 2, wherein the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.	Claim 13. The computer-readable storage device according to claim 11, wherein the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.
Claim 5. The computer-implemented method according to claim 4, wherein analyzing the executable file further comprises generating at least one	Claim 14. The computer-readable storage device according to claim 13, wherein analyzing the executable file further comprises generating a feature

feature vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.	vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.
---	---

363. None of the minor differences above have any material impact on my analysis. (The “the extracting of the plurality of static data points” is executed by the processor for the reason discussed above for Claim 10.) Accordingly, I incorporate by reference my analysis of Claims 2-5. Tahan in view of Menahem discloses each of Claims 11-14 based on the same analysis I performed for the corresponding Claim 2-5.

364. Thus, I conclude that Claims 11-14 are rendered obvious by Tahan in view of Menahem.

J.Claim 17 is rendered obvious by Tahan in view of Menahem

365. Claim 17 of the '844 Patent is nearly identical to Claim 1. I have visualized the differences between Claim 1 and Claim 17 below. Additions or changes to Claim 17 are in bold, and removed text is marked in strike-out.

Claim 1	Claim 17
[a] A computer-implemented method comprising:	[a] A system comprising:
	[b] at least one memory; and

	[c] at least one processor operatively connected with the memory and configured to perform operation of:
[b] extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file;	[d] extracting a plurality of predefined character strings from an executable file without decrypting or unpacking the executable file; ~~wherein the plurality of static data points represent predefined character strings in the executable file;~~
[c] generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static	[e] generating a feature vector from the plurality of predefined character strings using a classifier trained to classify the plurality of predefined character strings based on a collection of data comprising known malicious executable files, known benign executable files and known unwanted executable files, wherein the collection of data comprises at least a portion of one or more of the plurality of predefined character strings, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more

data points is within a predetermined range; and	predefined character strings from the plurality of predefined character strings is within a predetermined range; and
[d] evaluating the feature vector using support vector processing to determine whether the executable file is harmful.	[f] evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

366. Thus, as shown above, there are only a few differences between Claim 1 and Claim 17. I address these differences and my analysis of Claim 17 below.

1. Motivation to Combine

367. The motivation to combine Tahan and Menahem for Claim 17 is identical to the motivation to combine them for Claim 1 (and also discussed in Section IX.A.). I incorporate such analysis of the motivation to combine Tahan and Menahem into Claim 17 by reference.

2.Preamble

368. I understand that the preamble of a patent claim is not always considered limiting. However, to the extent that the preamble of Claim 17 of the '844 Patent is considered limiting, Tahan in view of Menahem discloses the preamble of Claim 17.

369. As I explained with reference to Claim 10, Tahan and Menahem both disclose algorithms implemented as instructions stored on a computer-readable storage medium and executed by a processor. A POSITA would have understood that such implementations teach the claimed “system”, especially in light of limitations 17[b] and 17[c], discussed hereinbelow.

3.Memory and Processor

370. Limitation 17[b] of the '844 Patent requires “at least one memory” and limitation 17[c] requires “at least one processor operatively connected with the memory and configured to perform operation of”.

371. Tahan discloses an implementation as computer programs (relating to its algorithms) executed on a processor. For example, Tahan discloses its setup. Tahan explains, “The evaluation computer used an Intel Q6850 CPU with 4GB of RAM. The processing time was measured using only 1 CPU core, although the implemented algorithm natively supported multiple cores.” (Ex. 1003, p. 962). A POSITA would have understood that running a program (e.g., relating to Tahan’s algorithms) on a computer with a processor (such as an Intel Q6850 CPU) involves a “processor operatively connected with the memory and configured to perform operation of” the program relating to Tahan’s algorithms/processes (e.g., whether the memory is being used to store the program and/or for other purposes relating to the overall process).

372. Menahem also discloses an implementation as a computer program executed on computers. Menahem discloses that its algorithms were “implemented in WEKA (Witten and Frank, 2005) in the JAVA programming language.” (Ex. 1004 at p. 1487). A POSITA would have understood that execution of a JAVA program involves both a memory and a processor operatively connected to the memory. Also, the processor would be configured to perform the steps of the computer program (e.g., relating to Menahem’s algorithms/processes).

373. Although Tahan discloses this element, a POSITA would have also looked to Menahem for additional details. I have already explained in section IX.A and Claim 1 why a POSITA would be motivated to combine Tahan and Menahem and I incorporate this analysis by reference.

4.Predefined Character Strings

374. The only other changes to the limitations of Claim 17 are replacing “static data points” with “predefined character strings.” However, this does not affect my analysis. As I discussed above with reference to Claim 1, static data points (e.g., disclosed by Tahan and Menahem) represent predefined character strings. Thus, in my opinion, static data points that represent predefined character strings (e.g., that are disclosed by Tahan and Menahem) disclose the claimed “predefined character strings” of Claim 17, or render them obvious. Accordingly, my analysis for Claim 1 is applicable to Claim 17 and I incorporate said analysis by reference.

5.All other elements of Claim 17

375. I incorporate all other analysis related to Claim 1 into my analysis of Claim 17 by reference. Tahan in view of Menahem discloses each of the remaining limitations of Claim 17 because of the same analysis for the corresponding limitation in Claim 1.

376. Thus, I conclude that Tahan in view of Menahem renders Claim 17 obvious.

K. Claim 19 is rendered obvious by Tahan in view of Menahem

377. Claim 19 of the '844 Patent is nearly identical to Claim 2. I have visualized the differences between Claim 2 and Claim 19 below. Additions or changes to Claim 19 are in bold, and removed text is marked in strike-out

Claim 2. The computer-implemented method according to claim 1, wherein the extracting the plurality of static data points further comprises classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data	Claim 19. The system according to claim 17, wherein the extracting the plurality of predefined character strings further comprises classifying the extracted plurality of predefined character strings according to a type of data extracted from
---	---

points for the classifier based on the classification.	the executable file, and encoding the extracted plurality of predefined character strings for the classifier based on the classification.
--	--

378. The only change to Claim 19 is replacing “static data points” with “predefined character strings.” However, this does not affect my analysis. As I discussed above with reference to Claims 1 and 2, static data points (e.g., disclosed by Tahan and Menahem) represent predefined character strings. Thus, in my opinion, static data points that represent predefined character strings (e.g., that are disclosed by Tahan and Menahem) disclose the claimed “predefined character strings” of Claim 19, or render them obvious. Accordingly, my analysis for Claim 2 is applicable to Claim 19 and I incorporate said analysis by reference. Tahan in view of Menahem discloses each limitation in Claim 19 based on the analysis for the corresponding limitation in Claim 2.

379. Thus, I conclude that Tahan in view of Menahem renders obvious Claim 19.

X. GROUND 2: CLAIMS 6-7, 9, 15-16, 18, AND 20 ARE RENDERED OBVIOUS BY TAHAN IN VIEW OF MENAHEM IN FURTHER VIEW OF SCHULTZ

380. As I describe in greater detail hereinbelow, a POSITA would have found claims 6-7, 9, 15-16, 18, and 20 rendered obvious by Tahan in view of Menahem in further view of Schultz.

A. Motivation to Combine Tahan and Menahem and Schultz

381. I have already explained with reference to Ground 1 that a POSITA would have been motivated to combine Tahan and Menahem. (See Section IX.A., Claim 1; see generally Section IX). I incorporate by reference my analysis of that motivation to combine into this section by reference.

382. A POSITA would have been further motivated to combine the teachings of Tahan and Menahem with those of Schultz. The Tahan and Menahem references are both research papers. Both disclose a range of ideas that are explored using research methods and evaluations. And while both references approach the innovations in terms of practical deployments, neither is a production system. Also, Tahan and Menahem ***both cite to Schultz as an author***. Thus, both were familiar with Schultz's work and considered it part of the "prior art."

383. Schultz, on the other hand, takes machine learning classifiers (such as those disclosed in Tahan and Menahem) and deploys them in a production system for blocking malicious email attachments. Schultz uses various machine learning (ML) classifiers and works with some of the same features, such as strings taken from PE headers (disclosed by Tahan and Menahem). (Ex. 1005, [0045]-[0055], [0068]; see also my analysis for Claim 1). Thus, a POSITA would have understood that Tahan, Menahem, and Schultz are in the same field and that their methods/systems are compatible. Like both Tahan and Menahem, Schultz recognizes that classifiers built from different algorithms can be used and substituted based on the type of problem being solved. A POSITA would have understood that the classifiers taught by Tahan and Menahem could be substituted for, modified by, and/or enhanced by the classifiers described in Schultz.

384. For example, Schultz provides very practical deployment instructions, such as only blocking an executable (e.g., identifying it as malware) if a score associated with the classification exceeds a threshold. (See my analysis for Claims 6 and 7 below). Schultz further emphasizes that this is used to better deal with false positives and to obtain a more favorable Receiver Operating Characteristic (ROC). (See *id.*). A POSITA would have recognized the importance of these goals and looked to Schultz for instructions on how to better deploy the teachings of Tahan and Menahem in practice.

B. Claim 6 is rendered obvious by Tahan in view of Menahem in further view of Schultz

385. Claim 6 of the '844 Patent is rendered obvious by Tahan in view of Menahem in further view of Schultz. This claim, which depends from Claim 1, reads:

The computer-implemented method according to claim 1, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.

386. This Claim depends from Claim 1. I have already explained how Tahan in view of Menahem render obvious Claim 1 in my analysis of Ground 1, and I incorporate that analysis by reference.

387. Schultz discloses preventing execution of an executable file when a determined probability value that the executable file is harmful exceeds a threshold. For example, Schultz discloses that executable files could be prevented from executing by quarantining them in the email system. “The results of this analysis are sent from parser3 to html-trap.procmail, which inserts warnings that the file may be malicious and may quarantine the attachment. The routine html-trap.procmail reintegrates filtered email back into normal email traffic.” (Ex. 1005, [0103]; see also *id.*, [0108]). A POSITA would have understood that the above disclosure’s “file” and “attachment” concern executable files because Shultz’s method/system concerns detecting and preventing malicious executable files, including malicious executable attachments: “A system and methods for detecting malicious executable attachments at an email processing application of a computer system using data mining techniques.” (Ex. 1005, [Abstract]). Also, a POSITA would have understood that quarantining an executable file results in preventing it from executing.

388. Schultz also discloses that the “analysis” used to quarantine email attachments (executable files) involves determining a probability value that the executable file exceeded a threshold. Schultz describes this feature thus:

A exemplary metric to identify borderline cases, which may be implemented in hexScan or a similar routine is to define ***a borderline case to be a case where the difference between the probability, or score, that the program is malicious and the probability, or score, it is benign is below a threshold.*** This threshold may be set based on the policies of the host. For example, in a secure setting, the threshold could be set relatively high, e.g., 20%. In this case, all binaries that have a **60/40** split are labeled as borderline. In other words, binaries with a **40-60%** chance (according to the model) of being malicious and **40-60%** chance of being benign would be labeled borderline. This setting can be determined by the system administrator. An exemplary default setting of **51.25/48.75** may be used with a threshold of 2.5%, which was derived from testing. (Ex. 1005, [0101], emphasis added).

389. In the above disclosure, Shultz shows that a probability that a program is malicious and a probability that a program is benign add to 100%. Thus, a POSITA would have understood that the disclosure that “a borderline case to be a case where the difference between the probability, or score, that the program is malicious and the probability, or score, it is benign is below a threshold” also discloses determining a program (executable file) to be malicious when “a determined probability value that the executable file is harmful exceeds a threshold value.” For example, in a situation where the above-discussed threshold value for a borderline case is set to 20%, anything below 60% for a probability that a program is malicious results in a borderline case (e.g., 59% (malicious) / 41% (benign) with 18 percentage point difference resulting in a “borderline” determination), or the program being determined to be benign. So, in that situation, the threshold probability value for a program (executable file) to be determined malicious would be 60% or higher. Thus, the above disclosure (along with other disclosures above) disclose “determining whether the executable file is harmful further comprises preventing execution of the

executable file when a determined probability value that the executable file is harmful exceeds a threshold value.” (See Ex. 1005, [0101]; see also *id.* [0022] (discussing a similar technique for “executable attachments”)).

390. Shultz also explains:

Advantageously, classifying the executable attachment may comprise predicting the classification of the executable attachment as one class in a set of classes consisting of malicious and benign. The set of classes may also include a borderline class. Classifying the executable attachment may comprise determining a probability or likelihood that the executable attachment is a member of each class in said set of classes based on said byte sequence feature. In one embodiment, this probability is determined by use of a Naive Bayes algorithm. In another embodiment, the probability may be determined by use of a Multi-Naive Bayes algorithm. The determination of the probability may be divided into a plurality of processing steps. These processing steps may then be performed in parallel. ***The executable attachment is classified as malicious if the probability that the executable attachment is malicious is greater than said probability that the executable attachment is benign.*** The executable attachment is classified as benign if the probability that the executable attachment is benign is greater than said probability that said executable attachment is malicious. ***The executable attachment is classified as borderline if a difference between the probability the executable attachment is benign and the probability the executable attachment is malicious is within a predetermined threshold.***

(Ex. 1005, [0022], emphasis added).

In accordance with the exemplary embodiment, a data mining-based filter integrates with Procmail's pre-existent security filter to detect malicious executable attachments. The filter uses a scoring system based on a data mining classifier to determine whether or not an attachment may be malicious. ***If an attachment's score is above a certain threshold it is considered malicious.*** The data mining classifier provides the ability to detect both the set of known malicious executables and a set of previously unseen, but similar malicious executables.

(Ex. 1005, [0038], emphasis added).

391. As shown above, Shultz explains that “The executable attachment is classified as malicious if the probability that the executable attachment is malicious is greater than said probability that the executable attachment is benign.” (Ex. 1005, [0022]). A POSITA would have

understood that the “probability that the executable attachment is benign” is a threshold value. Moreover, Shultz explains that “If an attachment’s score is above a certain threshold it is considered malicious.” (Ex. 1005, [0038]). Shultz uses the terms “score” and “probability” as related or interchangeable concepts: “A exemplary metric to identify borderline cases, which may be implemented in hexScan or a similar routine is to define a borderline case to be a case where the difference between the **probability, or score**, that the program is malicious and the **probability, or score**, it is benign is below a threshold.” (Ex. 1005, [0101], emphasis added). Thus, a POSITA would have understood that Shultz’s techniques could be implemented by using a score or a probability, so would have been motivated to apply a probability to Shultz’s disclosure at paragraph 38 such that “If an attachment’s score [or probability] is above a certain threshold it is considered malicious.” (Ex. 1005, [0038]).

392. Menahem teaches an ensemble classifier that outputs results that vary in “certainty.” For example, Menahem teaches a “Majority Voting” scheme that takes all of the outputs of the component classifiers as votes; a majority is sufficient for this scheme to classify the executable as either malicious or benign. (See Ex. 1004, p. 1485). However, the vote count also represents a confidence or probability score. Menahem also discloses “distribution summation” which is “to sum up the conditional probability vector obtained from each classifier (Clark and Boswell, 1991). The selected class is chosen according to the highest value in the total vector.” (*Id.*). If the probability of two classes (e.g., “benign” and “malicious”) are very close to each other, the confidence in the classification is decreased. In both of these two examples, there is a “score” associated with the result. A POSITA would have recognized that this score is compatible with Schultz’s threshold technique.

393. Shultz also discloses machine learning classifier algorithms that provide probability values regarding whether an executable file is harmful.

Advantageously, classifying the executable attachment may comprise predicting the classification of the executable attachment as one class in a set of classes consisting of malicious and benign. The set of classes may also include a borderline class. ***Classifying the executable attachment may comprise determining a probability or likelihood that the executable attachment is a member of each class*** in said set of classes based on said byte sequence feature. ***In one embodiment, this probability is determined by use of a Naive Bayes algorithm. In another embodiment, the probability may be determined by use of a Multi-Naive Bayes algorithm.*** The determination of the probability may be divided into a plurality of processing steps. These processing steps may then be performed in parallel. The executable attachment is classified as malicious if the probability that the executable attachment is malicious is greater than said probability that the executable attachment is benign. The executable attachment is classified as benign if the probability that the executable attachment is benign is greater than said probability that said executable attachment is malicious. The executable attachment is classified as borderline if a difference between the probability the executable attachment is benign and the probability the executable attachment is malicious is within a predetermined threshold. (Ex. 1005, [0022], emphasis added).

394. Thus, a POSTIA would have understood that Shultz's techniques are compatible with the Tahan-Menahem method/system. Moreover, I have already explained why a POSITA would have been motivated to combine Tahan and Menahem for Claim 1, from which Claim 6 depends. A POSITA would have been motivated to combine Schultz with Tahan and Menahem because Schultz provides very practical deployment instructions, such as only blocking an executable (e.g., identifying it as malware) if a score associated with the classification exceeds a threshold. Schultz further emphasizes that this is used to better deal with false positives and to obtain a more favorable Receiver Operating Characteristic (ROC). A POSITA would have recognized the importance of these goals and looked to Schultz for instructions on how to better deploy the teachings of Tahan and Menahem in practice. (See also Section X.A.)

395. Also, given that information on whether an executable file is harmful is used to prevent execution of the executable file (as discussed above), a POSITA would have understood that the Tahan-Menahem-Shultz method/system discloses or renders obvious “wherein determining whether the executable file is harmful **further comprises** preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.”

396. Thus, I conclude that Tahan in view of Menahem in further view of Schultz renders obvious this limitation.

C. Claim 7 is rendered obvious by Tahan in view of Menahem in further view of Schultz

397. Claim 7 of the '844 Patent is rendered obvious by Tahan in view of Menahem in further view of Schultz. This claim, which depends from Claim 6, reads:

The computer-implemented method according to claim 6, wherein the threshold value is set based on predetermined false positive range data.

398. This Claim depends from Claim 6. I have already explained how Tahan in view of Menahem in further view of Schultz renders obvious Claim 6.

399. Schultz discloses that the threshold used to block malware can be chosen based on pre-determined false positive rates. Schultz explains, and it was known in the art, that a “Receiver Operating Characteristic” (ROC) is a score derived from both false-positives and false-negatives. The ROC can be graphed illustrating the trade-off between false-positives (or false-negatives) and detection rates, because improving one generally makes the other worse. Schultz describes picking the threshold value based on these curves:

FIG. 11 displays the plot 400 of the detection rate 402 vs. false positive rate 404 using Receiver Operation Characteristic curves, as described in K. H. Zou et al., “Smooth Non-Parametric ROC Curves for Continuous Diagnostic Tests. Statistics in Medicine, 1997. Receiver Operating

Characteristic (ROC) curves are a way of visualizing the **trade-offs between detection and false positive rates**. In this instance, the ROC curve shows how the data mining method (illustrated in dashed line 405) can be configured for different environments. **For a false positive rate less than or equal to 1% the detection rate would be greater than 70%, and for a false positive rate greater than 8% the detection rate would be greater than 99%.** Thus, more secure settings would **select a threshold setting** associated with a point on the data mining line towards the right (indicated by arrow 408), and applications needing fewer false alarms should choose a point towards the left (indicated by arrow 406). (Ex. 1005, [0121], emphasis added).

400. Shultz explains that “The detection rate is the number of malicious binaries correctly classified divided by the total number of malicious programs tested.” (Ex. 1005, [0117]). A POSITA would have understood a more aggressive approach would improve the detection rate (e.g., and help identify more harmful executables) at an expense of a worsened false positive rate. Also, a POSITA would have understood that a more aggressive approach would set the above-discussed threshold value for a “probability value that the executable file is harmful” (of Claims 6 and 7) lower (e.g., which would help block harmful executables), but also at an expense of a worsened false positive rate. Thus, a POSITA would have understood that a false positive rate (or false positive range data) is an underlying factor/consideration in setting a malware detection method/system. Thus, a POSITA would have been motivated to use false positive range data in setting the above-discussed threshold such that “the threshold value is set based on predetermined false positive range data.”

401. I have already explained why a POSITA would have been motivated to combine Tahan and Menahem for Claim 1, from which Claims 6 and 7 depend. A POSITA would have been motivated to combine Schultz with Tahan and Menahem because Schultz provides very practical deployment instructions, such as only blocking an executable (e.g., identifying it as malware) if a score associated with the classification exceeds a threshold. Schultz further

emphasizes that this is used to better deal with false positives and to obtain a more favorable Receiver Operating Characteristic (ROC). A POSITA would have recognized the importance of these goals and looked to Schultz for instructions on how to better deploy the teachings of Tahan and Menahem in practice. (See also Section X.A.)

402. Thus, I conclude that Tahan in view of Menahem in further view of Schultz renders obvious Claim 7.

D. Claim 9 is rendered obvious by Tahan in view of Menahem in further view of Schultz

403. Claim 9 of the '844 Patent is rendered obvious by Tahan in view of Menahem in further view of Schultz. This claim, which depends from Claim 1, reads:

The computer-implemented method according to claim 1, wherein a determination of whether the executable file is harmful is used to retrain the classifier.

404. This claim depends from Claim 1. I have already explained how Tahan in view of Menahem renders obvious Claim 1 in my analysis of Ground 1.

405. Schultz discloses updating the classifier when the executable file was a borderline case but subsequently confirmed by a human expert.

A new classification rule set is generated at the model generator 240 by running the data mining algorithm on the new data set that contains the borderline cases along with their correct classification (as determined by expert analysis), and the existing training data set.
(Ex. 1005, [0110]).

406. The training examples of both Tahan and Menahem are compatible with this disclosure from Schultz. It would have been obvious to apply the retraining step from Schultz to Tahan and Menahem, as it would have been a known process with a known result.

407. I have already explained why a POSITA would have been motivated to combine Tahan and Menahem for Claim 1, from which Claim 9 depends. A POSITA would have been

motivated to combine Schultz with Tahan and Menahem because Schultz provides very practical deployment instructions, such as retraining the classifier. Although Tahan and Menahem did not explicitly disclose retraining, they do emphasize the importance of the training process. Tahan noted, for example, that “the accuracy [of the classifier] increases almost linearly as the training set size increases.” (Ex. 1003, p. 965). A POSITA would have recognized the importance of these goals and looked to Schultz for instructions on how to better deploy the teachings of Tahan and Menahem in practice. (See also Section X.A.)

408. Thus, I conclude that Tahan in view of Menahem in further view of Schultz renders obvious Claim 9.

E. Claims 15-16 are rendered obvious by Tahan in view of Menahem in further view of Schultz

409. Claims 15 through 16 are identical to a subset of Claims 6-7. In the table below, I visualize that these claims are identical except for the dependency and type of claim.

Claim 6. The computer-implemented method according to claim 1, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.	Claim 15. The computer-readable storage device according to claim 10, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.
Claim 7. The computer-implemented method according to claim 6,	Claim 16. The computer-readable storage device according to claim 15,

wherein the threshold value is set based on predetermined false positive range data.	wherein the threshold value is set based on predetermined false positive range data.
--	--

410. The main change to these claims is replacing “computer-implemented method” with “computer-readable storage device”. As I discussed in my analysis of Claim 10, a POSITA would have understood the algorithms taught by Tahan and Menahem to have been implemented as instructions stored in a computer-readable storage medium. A POSITA would have further found it obvious that Schultz’s teachings can be implemented as instructions stored in a computer-readable storage medium.

411. As the claims are effectively identical beyond that difference, I incorporate my analysis of Claims 6 and 7 by reference. Tahan in view of Menahem in further view of Schultz discloses or renders obvious these claims based on the analysis of the corresponding Claims 6 and 7. Thus, I conclude that Claims 15 and 16 are rendered obvious by Tahan in view of Menahem in further view of Schultz.

F. Claims 18, 20 are rendered obvious by Tahan in view of Menahem in further view of Schultz

412. Claims 18 and 20 are almost identical to a combination of Claim 6 and 7 and Claim 6 alone (respectively). In the table below, I visualize the differences between them with additions noted in bold and deletions in strikethrough.

Claim 6. The computer-implemented method according to claim 1, wherein determining whether the executable file is harmful further	Claim 18. The system according to claim 17, wherein the determining ~~whether the~~ executable file is harmful further
---	---

comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value. Claim 7. The computer-implemented method according to claim 6, wherein the threshold value is set based on predetermined false positive range data.	comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value, and wherein the threshold value is set based on predetermined false positive range data.
Claim 6. The computer-implemented method according to claim 1, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.	Claim 20. The system according to claim 17, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.

413. None of the changes are significant. Claim 18 is an almost identical incorporation of both Claim 6 and Claim 7; Claim 20 is effectively identical to Claim 6 alone. Accordingly, I incorporate my analysis of Claims 6 and 7 by reference for Claim 18 and my analysis of Claim 6

for Claim 20. Tahan in view of Menahem in further view of Schultz discloses Claim 18 based on the analysis of the Claim 6 and 7, and Claim 20 based on the analysis of Claim 6. Thus, I conclude that Claims 18 and 20 are rendered obvious by Tahan in view of Menahem in further view of Schultz.

XI. THERE ARE NO OBJECTIVE INDICIA OF NONOBVIOUSNESS

414. I am not aware of any objective indicia of non-obviousness related specifically to the '844 Patent. Indeed, I am not aware of any evidence of commercial success, long felt but unsolved need, failure of others, industry praise, or unexpected results specifically related to the '844 Patent.

XII. CONCLUSION

415. I declare that all statements made herein are true to the best of my knowledge and that all statements made on information and belief are believed to be true, and that this Declaration is made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of Title 18 of the United States Code.

Executed on March 3, 2023

By: 
Seth James Nielson