IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

_____

BEFORE THE PATENT TRIAL AND APPEAL BOARD

_____

TREND MICRO, INC.
Petitioner,

v.

WEBROOT, INC.
Patent Owner.

_____

IPR2023-01380

U.S. Patent No. 11,409,869

DECLARATION OF SETH JAMES NIELSON

TABLE OF CONTENTS

I. INTRODUCTION 1

II. QUALIFICATIONS 1

III. MATERIALS CONSIDERED 12

IV. LEGAL STANDARDS 14

V. Technical Background that Would Have Been Known to A POSITA 16

A. Machine Learning Techniques for Identifying Malware 17

B. Support Vector Machines 22

C. Clustering Models 24

D. Anti-Virus Scanning 25

E. Common Multiprocessing Configurations....26
F. Email Attachments....28
VI. OVERVIEW OF THE CHALLENGED PATENT....29
A. THE '869 PATENT SPECIFICATION AND CLAIMS....29
B. PROSECUTION HISTORY....35
VII. LEVEL OF SKILL IN THE ART....36
VIII. Overview of the Prior art....36
A. OVERVIEW OF "MAL-ID: AUTOMATIC MALWARE DETECTION USING COMMON SEGMENT ANALYSIS AND META-FEATURES" BY TAHAN ET AL. ("TAHAN")....36
B. OVERVIEW OF "IMPROVING MALWARE DETECTION BY APPLYING MULTI-INDUCER ENSEMBLE" BY MENAHEM ET AL. ("MENAHEM")....42
C. OVERVIEW OF US 2009/0254992 TO SCHULTZ ("SCHULTZ")....48
D. OVERVIEW OF US 8,885,928 TO FORMAN ("FORMAN")....49
IX. GROUND 1: Claims 1-4, 8, 10-25, 29, 32-39, 43, 46-53, and 57 are rendered obvious by TAHAN IN VIEW OF MENAHEM IN FURTHER VIEW OF Forman....54
A. Motivation to Combine Tahan, Menahem, and Forman....54
B. Claim 1 is rendered obvious by Tahan in view of Menahem in further view of Forman....64
1. Limitation 1[a]....64
2. Limitation 1[b]....65
3. Limitation 1[c]....68
4. Limitation 1[d]....70
5. Limitation 1[e]....71
6. Limitation 1[f]....82
7. Limitation 1[g]....108

C. Claim 2 is rendered obvious by Tahan in view of Menahem in further view of Forman 117

D. Claim 3 is rendered obvious by Tahan in view of Menahem in further view of Forman 119

E. Claim 4 is rendered obvious by Tahan in view of Menahem in further view of Schultz 122

F. Claim 8 is rendered obvious by Tahan in view of Menahem in further view of Forman 124

G. Claim 10 is rendered obvious by Tahan in view of Menahem in further view of Forman 128

1. Limitation 10[a] 130

2. Limitations 10[b]—10[g] 131

H. Claim 11 is rendered obvious by Tahan in view of Menahem in further view of Forman 131

I. Claim 12 is rendered obvious by Tahan in view of Menahem in further view of Forman 132

J. Claim 13 is rendered obvious by Tahan in view of Menahem in further view of Forman 132

K. Claim 14 is rendered obvious by Tahan in view of Menahem in further view of Forman 133

L. Claim 15 is rendered obvious by Tahan in view of Menahem in further view of Forman 133

1. Limitation 15[a] 135

2. Limitations 15[b]—15[g] 135

M. Claim 16 is rendered obvious by Tahan in view of Menahem in further view of Forman 136

N. Claim 17 is rendered obvious by Tahan in view of Menahem in further view of Forman 136

O. Claim 18 is rendered obvious by Tahan in view of Menahem in further view of Forman 136

1. Limitations 18[a]—18[f] 138

2. Limitation 18[g] 138

P. Claim 19 is rendered obvious by Tahan in view of Menahem in further view of Forman 141

Q. Claim 20 is rendered obvious by Tahan in view of Menahem in further view of Forman 142

R. Claim 21 is rendered obvious by Tahan in view of Menahem in further view of Forman 142

S. Claim 22 is rendered obvious by Tahan in view of Menahem in further view of Forman 143

T. Claim 23 is rendered obvious by Tahan in view of Menahem in further view of Forman 144

U. Claim 24 is rendered obvious by Tahan in view of Menahem in further view of Forman 145

V. Claim 25 is rendered obvious by Tahan in view of Menahem in further view of Forman 146

W. Claim 29 is rendered obvious by Tahan in view of Menahem in further view of Forman 147

X. Claims 32 and 46 are rendered obvious by Tahan in view of Menahem in further view of Forman 147

1. Limitations 32[a] and 46[a] 150

2. Limitations 32[b]—32[g] and limitations 46[b]—46[g] 150

Y. Claims 33-39, 43, 47-53, and 57 are rendered obvious by Tahan in view of Menahem in further view of Forman 151

X. GROUND 2: Claims 5-7, 9, 26-28, 30-31, 40-42, 44-45, 54-56, and 58-59 are rendered obvious by TAHAN IN VIEW OF MENAHEM IN FURTHER VIEW OF Forman in Further View of SCHULTZ 153

A. Motivation to Combine Tahan, Menahem, Forman, and Schultz 153

B. Claim 5 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 154

C. Claim 6 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 156

D. Claim 7 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 158

E. Claim 9 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 160

F. Claim 26 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 162

G. Claim 27 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 163

H. Claim 28 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 163

I. Claim 30 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 164

J. Claim 31 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 164

K. Claims 40-42, 44-45, 54-56, and 58-59 are rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz 166

XI. There are no objective indicia of nonobviousness 168

XII. Conclusion 168

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. 11,409,869 ("the '869 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 '869 patent are obvious in light of the prior art discussed below. The '869 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 '869 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 1-4, 8, 10-25, 29, 32-39, 43, 46-53, and 57 are rendered obvious by the combination of Tahan, Menahem, and Forman

b. Claims 5-7, 9, 26-28, 30-31, 40-42, 44-45, 54-56, and 58-59 are rendered obvious by the combination of Tahan, Menahem, Forman, and Schultz

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 1004**. 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 '869 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. 11,409,869 to Schmidtler et al. ('869 patent)
- Ex. 1002: Prosecution history of the '869 patent
- Ex. 1003: Expert Declaration of Dr. Seth James Nielson
- Ex. 1004: Curriculum Vitae of Dr. Seth James Nielson
- Ex. 1005: 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. 1006: Eitan Menahem, Improving Malware Detection by Applying Multi-Inducer Ensemble, Computational Statistics & Data Analysis 53, (2009). ("Menahem")
- Ex. 1007: U.S. Published Patent Application No. 2009/0254992 to Schultz et al. Systems and Methods for Detection of New Malicious Executables. ("Schultz")

- Ex. 1008: Stuart Russell, Artificial Intelligence: A Modern Approach; The Intelligent Agent Book, 749-751, (Prentice Hall 2) (2003).
- Ex. 1009: Lars Buitinck, API design for machine learning software: Experiences from the scikit-learn project, (September 2013).
- Ex. 1010: Declaration of Ingrid Hsieh-Yee, Ph.D.
- Ex. 1011: Dr. Solomon's Anti-Virus Toolkit Reference Guide (April 1999).
- Ex. 1012: George Karypis, CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling, IEEE 32 (8), (August 1999).
- Ex. 1013: Yingjie Tian, Recent Advances on Support Vector Machine Research, Technological and Economic Development of Economy 18(1), (April 2012).
- Ex. 1014: Ian H. Witten and Eibe Frank, Data Mining, Practical Machine Learning Tools and Techniques 2, (July 2005).
- Ex. 1015: Craig Szydlowski, Multithreaded Technology & Multicore Processors, Dr. Dobb's Journal, (May 2005).
- Ex. 1016: Webroot, Inc. and Open Text, Inc. v. Trend Micro Inc., Case No. 6:22-cv-00243-ADA-DTG, Claim Construction Order, Dkt. 391 (W.D. Tex., July 25, 2023).
- Ex. 1017: U.S. Patent No. 7,657,838 to Daniell et al.
- Ex. 1018: U.S. Patent No. 8,885,928 to Forman ("Forman")
- Ex. 1019: U.S. Published Patent Application No. 2010/0082642 to Forman et al. ("'642 publication")

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 ’869 patent is arguing that the ’869 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; I 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 '869 patent that would have been known to a POSITA as of the filing date of the '869 patent, which I understand to be February 14, 2020. Any discussion of the background technologies in this section is 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; Dikinson, 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. 1006, p. 1).

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 '869 patent was filed. (*See* Ex. 1005, p. 3). 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. 1005, pp. 2-3).

65. The output of the first phase is a vector of all the features identified and extracted from the training data. However, not all the identified features may be useful or necessary for classification.

66. The purpose of the second phase, feature selection, is to reduce the number of identified or extracted features to only the ones necessary to classify correctly, in order to reduce computational complexity:

> In ML applications throughout numerous application domains the large number of features, many of which not contributing to the classification accuracy (or even diminishing it), presents a huge problem. Thus, reducing the number of features is crucial, and must be performed while maintaining a high level of accuracy. This is due to the fact that the vocabulary may exceed millions of features, far more than can be processed by any classification algorithm within a reasonable period of time. Additionally, it is important to identify those terms that appear in most of the files, in order to avoid vectors that contain many zeros. Feature selection methods aim to reduce the dimensionality and enhance compactness of the feature vector representing the files.
> (Shabtai, p. 18).

67. 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)).

68. 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. 1006, p. 2).

69. Once the classifier is created, it generally takes an unknown sample as input and spits out a classification. This input can be any file, whether seen previously or not. The only requirement for attempting to classify the input is that the features used by the classifier can be extracted from the input and provided to the classifier.

70. In addition, 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. 1007, [0110]).

71. 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 identified and extracted for both ***training*** the classifier, and for ***classifying*** the unknown samples. In addition, feature selection or scaling may be performed during training and while classifying unknown samples. (Ex. 1018, 7:60-8:13). 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 identifies and 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 (perhaps after narrowing or scaling the features) into the (trained) classifier

e. Fifth, the classifier outputs a label (e.g., "malicious" or "benign") or a probability 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.

72. The importance of the retraining step was well known to practitioners in the art. For example:

> A major challenge facing the anti-virus community is the dynamic nature of malware and the need for constant rapid acquisition of new malware samples in order to be aligned with emerging threats. ***The acquisition of new malicious files is essential for updating the classifiers (re-training).*** This is often done by using honey-pots or by continuous scanning of network traffic in an attempt to detect a new malware. However, the main challenge in both options is to scan all collected files efficiently, especially when scanning high-throughput networks operating at multi-GB/s rates. ***In addition manual inspection of potentially malicious files is time-consuming, but in some "hard" cases need to be performed by human experts.***
> (Shabtai, pp. 26-27, emphasis added).

73. It should be noted that the retraining may use files that have been classified by the classifier. However, the label for any data used for training or retraining should be assigned independently of the classifier's predictions in order to ensure high confidence in the accuracy of

the label. In other words, if the classifier says that a new file is "malicious", this does not convey sufficient confidence 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. This label can be safely used for training/re-training.

74. The data used to train a model is usually labeled. That is, the correct classification for each sample is known beforehand. The data set may be subdivided into groups for specific purposes, such as training, validation, and testing. The process of creating a classifier is often iterative and may involve re-sampling the data. Witten explains it this way:

> It is important that the test data was not used *in any way* to create the classifier. For example, some learning methods involve two stages, one to come up with a basic structure and the second to optimize parameters involved in that structure, and separate sets of data may be needed in the two stages. Or you might try out several learning schemes on the training data and then evaluate them—on a fresh dataset, of course—to see which one works best. But none of this data may be used to determine an estimate of the future error rate. In such situations people often talk about three datasets: the *training* data, the *validation* data, and the *test* data. The training data is used by one or more learning methods to come up with classifiers. The validation data is used to optimize parameters of those classifiers, or to select a particular one. Then the test data is used to calculate the error rate of the final, optimized, method. Each of the three sets must be chosen independently: the validation set must be different from the training set to obtain good performance in the optimization or selection stage, and the test set must be different from both to obtain a reliable estimate of the true error rate.
>
> It may be that once the error rate has been determined, the test data is bundled back into the training data to produce a new classifier for actual use. There is nothing wrong with this: it is just a way of maximizing the amount of data used to generate the classifier that will actually be employed in practice. What is important is that error rates are not quoted based on any of this data. Also, once the validation data has been used—maybe to determine the best type of learning scheme to use—then it can be bundled back into the training data to retrain that learning scheme, maximizing the use of data.
>
> (Ex. 1014, pp. 9-10, italics in original).

75. It is also worth mentioning that because a classifier (or model) is constructed from a specific machine learning algorithm, it is not unusual to identify it as that "type" of classifier (or model). So, for example, a classifier trained by Support Vector Machines (SVM) techniques is often called an SVM Classifier (or Model). (*See, e.g.*, Ex. 1006 generally).

B. 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. 1008), reproduced hereinbelow:

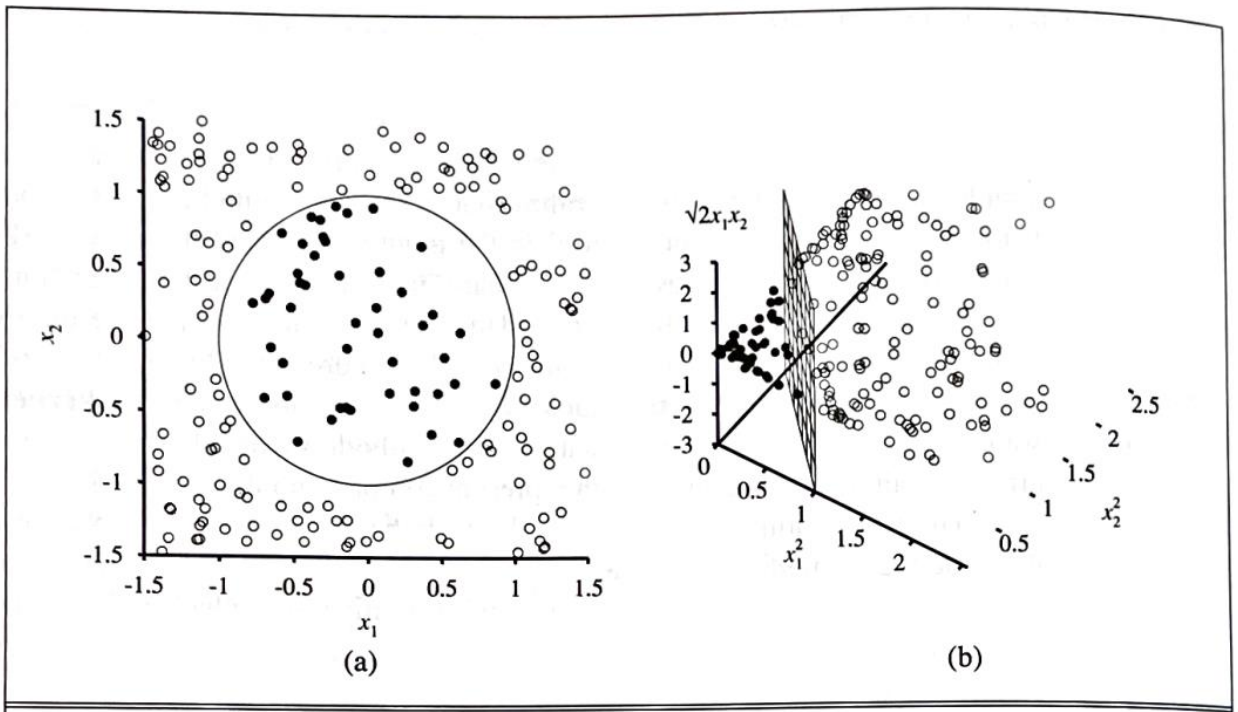


Figure 20.27 (a) A two-dimensional training set with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 \leq 1$, is also shown. (b) The same data after mapping into a three-dimensional input space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. The circular decision boundary in (a) becomes a linear decision boundary in three dimensions.

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. 1008, p. 3).

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.*, p. 3). 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.*, pp. 3-5).

80. In addition, SVM can produce probabilistic outputs instead of strict categorical classifications. For example, Tian et al. in "Recent Advances on Support Vector Machines Research" describe one technique for obtaining such probabilistic outputs from an SVM:

> For a binary classification problem with the training set (1), standard C-SVM computes a decision function (2) such that it can be used to predict the label of any test input x. However, we cannot guarantee that the deduction is absolutely correct. ***So sometimes we hope to know how much confidence we have, i.e. the probability of the input x belonging to the positive class.*** To answer this question, investigate the information contained in g(x). It is not difficult to imagine that the larger g(x) is, the larger the probability is. So the value of g(x) can be used to estimate the probability $P(y = 1 \mid g(x))$ of the input x belonging to the positive class.
> (Ex. 1013, p. 21, emphasis added).

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

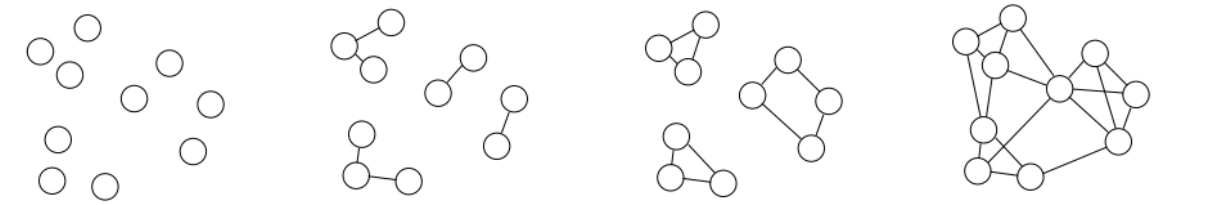
C. Clustering Models

82. "Clustering in data mining is a discovery process that groups a set of data such that intracluster similarity is maximized and intercluster similarity is minimized." (Ex. 1012, p. 1). Many algorithms exist to perform such clustering. For example, some, such as "K-means, PAM, CLARANS, DBSCAN, CURE, and ROCK", operate by accepting static parameters and discovering clusters among the data based on those parameters, which can find clusters of homogeneous "shapes, densities, and sizes." (*Id.*). Other algorithms, such as CHAMELEON, operate in multiple steps first to quantify the relatedness of data points (e.g., by constructing sparse sub-graphs) and then to discover clusters based on those sub-graphs, even if the clusters are "of different shapes, densities, sizes, noise, and artifacts." (*Id.*).

83. K-nearest neighbor (KNN) is a technique for constructing graphs necessary for a clustering model such as CHAMELEON:

> CHAMELEON's sparse graph representation of the data items is based on the commonly used k-nearest neighbor graph approach. Each vertex of the k-nearest neighbor graph represents a data item, and there exists an edge between two vertices, if data items corresponding to either of the nodes is among the k-most similar data points of the data point corresponding to the other node. Figure 7 illustrates the 1-, 2-, and 3-nearest neighbor graphs of a simple data set. Note that since CHAMELEON operates on a sparse graph, each cluster is nothing more than a sub-graph of the original sparse graph representation of the data set.
> (*Id.*, p. 7).

84. Figure 7 of Karypis illustrates the output described:



(a) Original Data in 2D (b) 1-nearest neighbor graph (c) 2-nearest neighbor graph (d) 3-nearest neighbor graph

Figure 7: *k*-nearest graphs from an original data in 2D.

(*Id.*, p. 8).

85. The algorithm then uses the k-nearest neighbor graph for the next step of discovering clusters: "CHAMELEON finds the initial sub-clusters using a graph partitioning algorithm to partition the k-nearest neighbor graph of the data set into a large number of partitions such that the edge-cut, i.e., the sum of the weight of the edges that straddle partitions, is minimized." (*Id.*, p. 10).

D. Anti-Virus Scanning

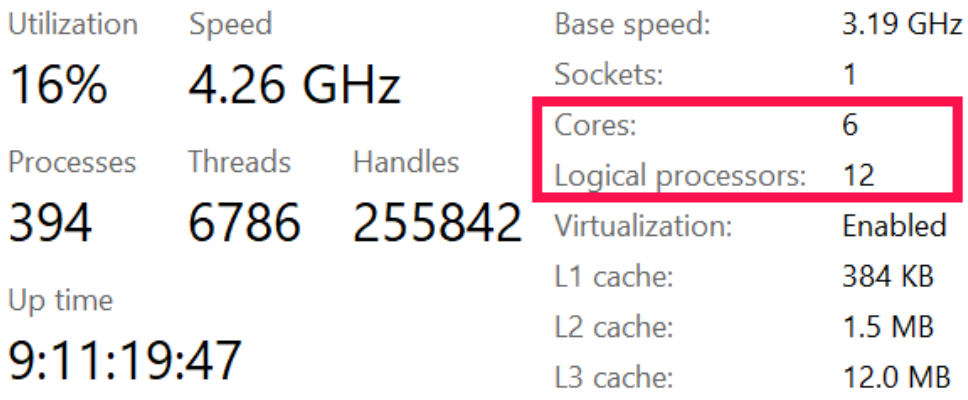
86. Anti-virus software can operate in various modes. One mode is performing a full scan of the filesystem. This can be performed on demand or on a schedule (e.g., when users are less likely to need the machine). (See Ex. 1011, p. 3).

87. Another mode is to scan when files are accessed (e.g., opened, or attempted to execute). For example, the user guide from one particular anti-virus software available in 1999, Dr. Solomon's Anti-Virus Toolkit, describes such "on-access" scanning:

> PCs [stand-alone or workstations] should be protected with an **on-access scanner** [VirusGuard and/or WinGuard], to provide the first layer of protection 'in-depth' [rather than at the perimeter]. The on-access scanner will scan disks and files before they are used. The on-access scanner runs in the background [requiring no action on the part of the user]. The user will be given a pop-up warning, to identify the virus; and the user will not be able to use the infected disk or file. VirusGuard and WinGuard provide protection for floppy disks, local hard disks and network drives. They are fully-configurable, to enable greater or lesser security [for example, checking files which are written to disk may be selected for those PCs which are downloading software, documents, etc. from a remote location [the Internet, **BBS**, etc.]]. WinGuard may be configured to auto-disinfect, so that disks and files may be cleaned automatically, on detection. This makes anti-virus management easier [virus removal is carried out automatically, rather than by a member of the IT Department]. WinGuard may be configured to log all virus incidents, allowing the IT Department to monitor all virus incidents.
> (Ex. 1011, p. 2, emphasis and square brackets in original).

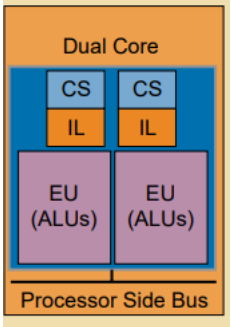
E. Common Multiprocessing Configurations

88. Although it is typical to refer to a computer's processor (singular), most processors now and at the time of the alleged invention are actually a type of multiprocessor. For example, on my current primary desktop machine the single processor is actually treated like ***12*** processors ("logical processors") using a technique called "cores."



89. In 2005, it was acknowledged that there was a "transition of CPUs from single to multicore implementations." (Ex. 1015, p. 1). But even before multicore, Intel had produced processors that employed "Hyperthreading" wherein "the operating system recognizes two logical processors and schedules tasks to each logical processor." (*Id.*, p. 2). Thus, by 2005, persons of skill in the art would have known the concept of multiple logical processors and that multicore technology was making logical processors more powerful.

90. The concept of a multicore processor is one in which "[e]ach core contains its own dedicated processing resources similar to an individual CPU, except for the Processor Side Bus, which may be shared between these two cores." *(Id.*, p. 2). The following illustration illustrates this architecture for two cores. Each core is shown with its own ALU (arithmetic logic unit), CPU state (CS), and interrupt logic (IL).



(*Id.*, Figure 1).

91. Thus, a POSITA would have recognized that even single computers with supposedly single processors are often multicore systems that are designed like, and behave like, multi-processor systems.

92. A POSITA also would have understood that for multicore systems, like any multiprocessor system, to be effective the software must be ***parallelized***. This means that there must be two parts of the software that can run at the same time. This can be done by having two separate programs each running on a different core or by having a single program with ***threads***.

"[Multicore] CPU implementations require threaded code to fully employ their computing potential… You should choose a threaded programing model that suits the parallelism inherent to the application." (*Id.*, p. 2). One way to parallelize software is to decompose the functionality into independent operations. "When there are a number of independent tasks that run in parallel, the application is suited to functional decomposition… To exploit multicore CPUs, you identify the parallelism within your programs and create threads to run multiple tasks concurrently." (*Id.*).

F. Email Attachments

93. Email messages can be sent in various formats, some of which allow for "self-executing code." The following description is from U.S. Patent No. 7,657,838 to Daniell et al., filed in 2003 and issued in 2010:

> Modern email systems are amenable to email messages in either a plain-text format (e.g., American standard code for information interchange (ASCII)) or an encoded format (e.g., hyper-text markup language (HTML), multi-purpose Internet mail extension (MIME), etc.). While plain-text email messages are sufficient for simply conveying message content to intended recipients, these plain-text email messages provide very few options on how the email messages are displayed due to their limited functionality.
>
> Conversely, email messages in encoded formats have greater functionality. For example, HTML-formatted email messages may include bitmap images that are directly embedded into the email message itself. Thus, when a recipient receives the HTML-formatted email message, the embedded bitmap image is automatically displayed to the recipient in an email read window.
>
> While many of the embedded items in encoded-format email messages are harmless, ***the ability to introduce self-executing code into encoded-format email messages results in a greater susceptibility of encoded-format email messages to viruses, worms, Trojan programs, etc.***
>
> Thus, a heretofore unaddressed need exists in the industry to reduce the susceptibility of encoded-format email messages to undesired executable programs.
> (Ex. 1017, 1:20-44, emphasis added).

94. Given the cybersecurity threat posed by such "self-executing code," Daniell proposes systems and methods for preventing such execution by detecting content that may attempt to execute and removing it:

> In a broad sense, the several embodiments disclosed herein provide systems and methods that are more secure from attack by viruses, worms, or Trojan programs. Since these undesired programs may be embedded as self-executing code in non-ASCII-based email systems (e.g., HTML-based email systems), ***the generation of a modified email message without the embedded functions, which are amenable to the embedding of self-executing codes, may greatly reduce the opportunity for such attacks.***
> (Ex. 1017, 2:42-50, emphasis added).

95. Without the intervention of such systems, malicious content would execute on the target machine (e.g., an email client):

> Thus, if the user chooses to view the entire contents of the original email message, including the embedded functions, the user may open the saved message using a conventional email client. For those instances, ***if the original email message is corrupted by a harmful self-executing code, such as a virus, worm, or Trojan program, then the conventional email client would typically launch that self-executing code.***
> (Ex. 1017, 5:32-39, emphasis added).

VI. OVERVIEW OF THE CHALLENGED PATENT

A. THE '869 PATENT SPECIFICATION AND CLAIMS

96. U.S. Patent No. 11,409,869 ("the '869 patent") relates to detection of malware in executable files. (Ex. 1001 at Abstract). The '869 patent emphasizes that it analyzes previously unknown executables. (*Id.*, 1:7-10). The alleged invention of the '869 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.

(*Id.*, 2:12-22).

97. Within the specification, the '869 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 '869 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 (*Id.*, 8:8-34).

b. Narrowing features used (*Id.*, 12:61-13:3).

c. Training a classifier (*Id.*, 9:27-54).

98. The classifiers disclosed in the '869 include SVM-trained classifiers and combination methods wherein multiple classifiers are combined into an ensemble classifier. (*Id.*, 9:27-54). 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 '869 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. (*Id.*, 9:27-54). The patent also teaches that many other techniques besides SVM are appropriate for this type of problem and anticipates their use by a POSITA, including "an artificial neural network (ANN), a decision tree, association rules, inductive logic, a support vector machine, clustering analysis, and Bayesian networks, among other examples." (*Id.*, 7:64-8:8).

99. The '869 patent discloses a "knowledge component 102" that has broad functionality during several steps of the approach. The following are its functions:

a. "[c]ollect data for use in building, training and/or re-training a learning classifier to evaluate executable files" (*Id.*, 3:54-57).

b. "[P]rovide[]" information to the classifier, including to "train/re-train" it (*Id.*, 7:26-42, 9:31-34).

c. Interface with the "threat detection component 106" "for training/re-training associated with the learning classifier" (*Id.*, 10:58-63).

d. "[I]dentify static data points for collection" and "associate the static data points with one of the categories of files" (*Id.*, 5:11-15).

e. "[C]ollect file data", "parse the file data, and store extracted data points" (*Id.*, 5:15-52), including "continuously collect[ing] and manag[ing] data" (*Id.*, 3:57-60).

f. "[M]anage[]" stored data (*Id.*, 7:21-25).

g. "[E]valuate an executable file and extract/collect static data points for evaluation of the file by the learning classifier" (*Id.*, 7:1-7).

100. In addition, the '869 patent discloses the following:

> The knowledge component 102 ***is one or more storages, memories, and/or modules*** that continuously collects and manages data that may be used to detect threats in files such as executable files. In one example, the knowledge component 102 maintains a robust collection of data comprising known malicious executable files, known benign executable files, and potentially unwanted executable files.
> (*Id.*, 3:57-63, emphasis added).

101. The specification later explains that the knowledge component may include programs in addition to data: "For example, the knowledge component 102 may comprise one or more storages such as databases, and one or more additional components (e.g., processors

executing programs, applications, application programming interfaces (APIs), etc.)." (*Id.*, 5:47-52).

102. In addition, the specification describes a "knowledgebase" or "knowledge base" as either a component or an output of the "knowledge component 102", which may train or be used to train a classifier. (*Id.*, 4:39-43, emphasis added: "a knowledgebase that may ***be used to periodically train*** a classifier"; *Id.*, 11:60-12:4, emphasis added: "The knowledge base may ***be used to automatically train/re-train*** one or more learning classifiers"; *Id.*, 12:55-60, emphasis added: "the learning classifier ***trained by*** the knowledge base"). Furthermore, "[d]ata maintained by the knowledge component 102 may be continuously updated" to keep the system "up-to-date on current threats." (*Id.*, 4:51-5:7). In summary, the knowledge component includes (or is) data storage, and it also performs many functions related to collecting, storing, managing, and providing data necessary for training and classifying.

103. The '869 patent discusses generating feature vectors from static data points. (*Id.*, 5:53-6:18, 8:21-9:26). The patent discusses, as an example, PE header data and n-grams as different examples of static data points. (*See, e.g., id.*, Table 1.1, 5:57-6:18). One claim of the '869 patent recites that the static data points represent "predefined character strings." (*See, e.g., id.*, Claim 8).

104. 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 '869 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."

105. For example, the '869 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." (*Id.*, 6:50-52, 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." (*Id.*, 7:64-67). In fact, the specification states that, "a learning classifier [is] built by the learning classifier component 102." (*Id.*, 7:18-19). 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." (*Id.*, 7:32-39).

106. The '869 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." (*Id.*, 12:55-63, 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).

107. For example, the *Scikit-Learn* project was an open source machine learning library for Python, which I have used myself. (*See* Ex. 1009). 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. 1009, p. 5, emphasis added).

108. 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 '869 patent. (*See id.*).

B. PROSECUTION HISTORY

109. Prosecution on the ’869 patent began with a non-final rejection on March 29, 2021 in which the Examiner raised rejections under 35 U.S.C. 112, 35 U.S.C. 102/103, and double patenting and rejected all of claims 2-21 – claim 1 was cancelled in a preliminary amendment. The Examiner rejected claims 2-5, 9, 11-16, 18-20 under 35 U.S.C. 102(a)(1) based on Shabtai et al. (“Detection of malicious code by applying machine learning classifiers on static features…”); claims 6-8 were rejected 35 U.S.C. 103 under Shabtai and US 2010/0293273 to Basarrate et al.; and claims 10, 17, and 21 were indicated as allowable. (Ex. 1002, pp. 231-236). After an examiner interview, Patent Owner amended the independent claims to claim subject matter broader than that recited in allowable claim 10. (Ex. 1002, pp. 212-219). The Examiner then allowed the application in a Notice of Allowance on August 18, 2021. (Ex. 1002, pp. 183-184).

110. Patent Owner then filed an RCE on November 18, 2021 with new amendments adding claims 22-51 and removing “by a classifier” from the independent claims. (*Id.*, pp. 163-170). The Examiner then issued a non-final office action rejecting claims 2-5, 9, 11-16, 18-20, 22-28, 33, 35, 37-43, 48, and 50 under 35 U.S.C. 103 based on Shabtai and US 2010/0082642 to Forman et al. (the “’642 publication”; Ex. 1019)[1], while claim 10 and related claims were again indicated as allowable. (*Id.*, pp. 136-142). Patent Owner then amended independent claims 2, 12, 18, 22, and 37 to recite the allowable subject matter of claim 10 without rebutting the Examiner’s assertion that the ’642 publication discloses turning features on or off for classification, effectively conceding that the ’642 publication teaches this claim limitation. (*Id.*, pp. 120-125). The Examiner then allowed the application on 4/15/2022. (*Id.*, pp. 81-82).

[1] The ’642 publication asserted by the Examiner during prosecution is not the same as Forman asserted in Grounds 1 and 2 (i.e., U.S. Patent No. 8,885,928; Ex. 1018). Forman discloses details discussed in the Grounds not included in the ’642 publication. However, both Forman and the ’642 publication disclose turning features on or off for classification.

111. The Patent Owner then filed an RCE on May 10, 2022 making further amendments adding claims 52-65 reciting: "wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range". (*Id.*, pp. 57-65). The Examiner then allowed the application along with the new claims on June 15, 2022. (*Id.*, pp. 23-26). The '869 patent then issued on August 9, 2022.

VII. LEVEL OF SKILL IN THE ART

112. 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 '869 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.

113. 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")

114. 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. 1005, p. 1).

115. 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. 1005, pp. 2-3).

116. 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.

> 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. 1005, p. 2).

117. 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. 1005, pp. 2-3, emphasis added).

118. 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. 1005, pp. 2-3).

119. 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.

120. 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. 1005, p. 1). As stated, Tahan identifies improvements to the art in terms of features and feature selection: meta-features and common segment analysis.

121. Tahan also discloses analyzing segments (e.g., "n-gram—segments of consecutive bytes from different locations within the executables of length n. Each n-gram extracted is considered a feature", Ex. 1005, p. 3) 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. 1005, p. 4).

122. 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. 1005, p. 7). 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.*).

123. 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. 1005, p. 10). On the other hand, "segments that actually reside in the malware are left in the segment collection." (Ex. 1005, p. 11).

124. 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. 1005, p. 27).

125. 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.

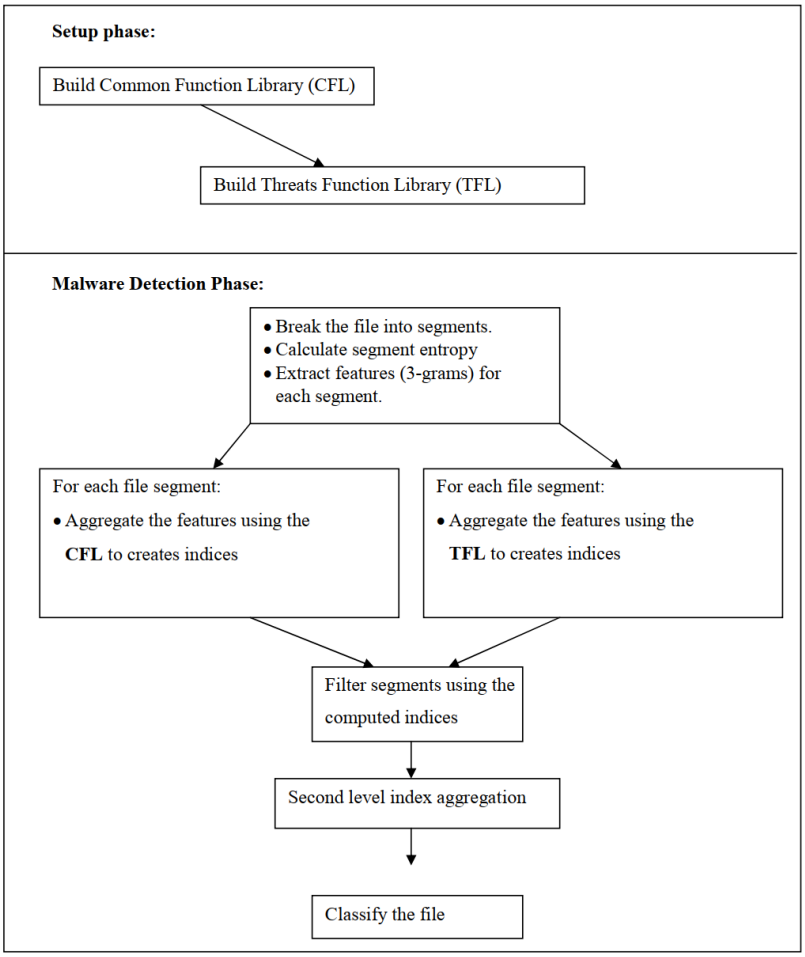


Figure 1: The Mal-ID method for detecting new malware applications.

126. 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. 1005, p. 9). 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. 1005, p. 10). 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.

127. 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. 1005, p. 11). Specifically, Tahan combines Mal-ID with three other classification technologies known as "Rotation Forest," "C4.5", and a "multilayer perception." (Ex. 1005, p. 13). 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. 1005, p. 28).

128. 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. 1005, pp. 3-5). 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 '869 patent. (*See id.*)

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

129. 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 journal "Computational Statistics and Data Analysis" on October 19, 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. 1006, p. 1).

130. 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. 1006, p. 5, emphasis added).

131. 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. 1006, p. 5).

132. 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. 1006, pp. 5-6).

133. 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. 1006, p. 6). Using these function marks, Menahem extracts the following attributes/features, which are derived from an executable file:

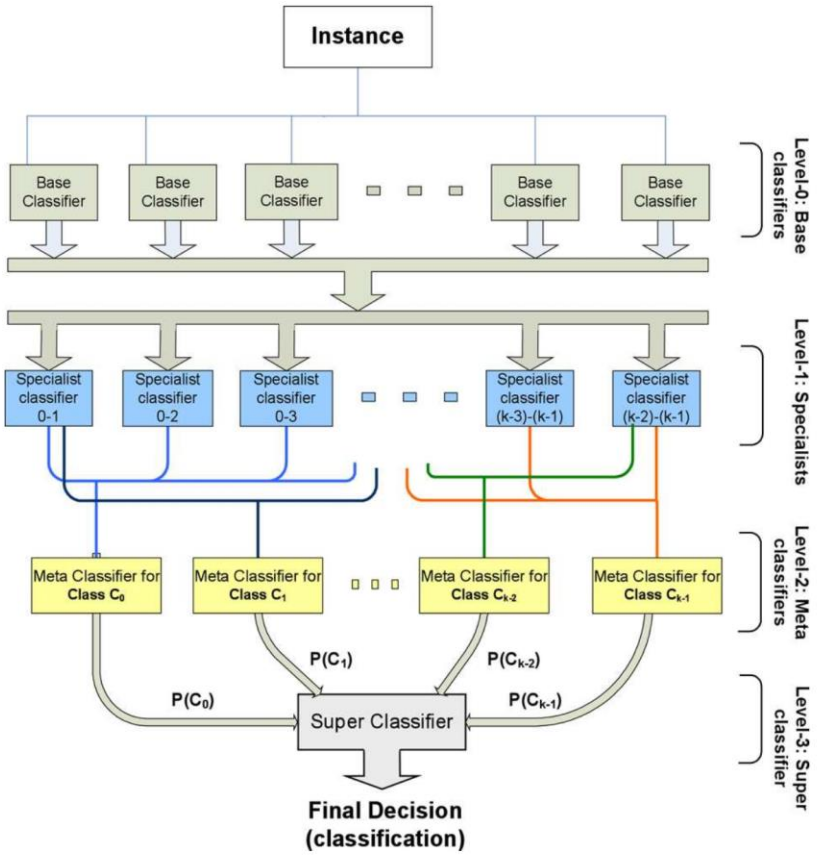
- Size of file
- File's entropy value
- Total number of detected functions
- Size of the longest function that was detected
- Size of the shortest function that was detected
- Average size of detected functions
- Standard deviation of the size of detected functions
- Number of functions divided into 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. 1006, p. 6).

134. 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. 1006, p. 2). In this example from Menahem, the ensemble of combined classifiers (such as SVM and other classifiers) was 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. 1006, p. 3).

135. 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. 1006, p. 11). 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. 1006, p. 4).

136. 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. 1006, p. 5). 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. 1006, p. 12).

137. 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. 1006, p. 2).

138. Menahem further discloses that classifiers could be trained on 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. 1006, p. 12). 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. 1006, p. 2).

139. 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. 1006, p. 12, emphasis added).

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

140. 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." (Ex. 1007, 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.*).

141. Schultz teaches identifying potentially malicious email attachments (e.g., executable files) before they can be executed by users and classifying the attachments as malicious or benign:

> The process 100 of detecting malicious emails in accordance with the invention is illustrated in FIG.8. ***A first step is to receive the emails at the server (step 102).*** In the exemplary embodiment, the mail server is Sendmail. Procmail is a publicly available program that processes e-mail messages received by the server and looks for particular information in the headers or body of the message, and takes action on what it finds.
>
> ***Subsequently, the emails are filtered to extract attachments or other components from the email (step 104).*** The executable attachments may then be saved to a file. In the exemplary embodiment, html_trap.procmail, a commonly available routine, has been modified to include a call to a novel routine, parser3, which performs the functions of filtering attachments. The routine parser3 includes a call to the routine extractAttachments, for example, which extracts executable attachments and other items of the email and saves them to a file…
> (Ex. 1007, [0096]-[0097], emphasis added).
>
> ***Features in the executable attachment are extracted (step 106), and those features are subsequently analyzed and used to classify the executable attachment as malicious or benign.***
> (Ex. 1007, [0098], emphasis added).

142. 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. 1007, [0045]-[0055], [0068]).

143. 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. 1007, [0110]).

D. OVERVIEW OF US 8,885,928 TO FORMAN ("FORMAN")

144. U.S. Patent No. 8,885,928 to Forman is directed to "machine-learning classification… [where] [t]he values for the features are scaled according to the scores for the features, and the item is classified by inputting the adjusted feature set values for the item into a previously trained classifier." (Ex. 1018, Abstract).

145. In terms of machine-learning technique, Forman identifies multiple approaches including Support Vector Machines (SVM) as an example of supervised learning. As Forman explains, "In the preferred embodiments of the invention, training section 16 implements Support Vector Machine (SVM) training in order to output its classifier 18." (Ex. 1018, 3:46-49). Additionally, Forman notes that binary classifiers, like SVM, are often extended to multi-classification (classifying into more than two labels) using multiple classifiers. A single classification of the many (1-of-n) can be determined "by decomposing the 1-of-n problem, pitting each class against the others." (Ex. 1018, 2:23-24). Alternatively, it is possible to identify many matching classes out of the set of total classes (n-of-m) "by applying m independent binary classifiers to each item." (Ex. 1018, 2:26).

146. In identifying the feature set for a classifier or (or feature sets for a collection of classifiers), Forman notes that it often includes features that have little value. As Forman explains:

> It often will be the case that the number of features included in the original feature set is over-inclusive, e.g., including some features that are not very predictive of the desired classification category. There are several reasons that this situation can occur. First, one often wants to increase the likelihood that the feature set is able to adequately characterize the input items, and at the outset it sometimes is not possible to know which features will prove to be most predictive, so it often is better to err on the side of over-inclusion. In addition, in order to minimize processing requirements, the same feature set sometimes will be generated and used for multiple different purposes, so the resulting feature set has some features that are not particularly appropriate for the specific task at hand.
> (Ex. 1018 2:59-3:5).

147. These same concepts are discussed in Tahan and Menahem in terms of *feature selection* during training. As discussed above, Tahan and Menahem both reduce features for training in order to eliminate unnecessary features and/or reduce the processing load. Forman also suggests reducing features during training. (Ex. 1018, 3:6-55).

148. However, Forman also discloses feature selection, as part of a more general process called *feature scaling*, during classification as well as during training. Feature selection involves using weights to reduce the impact of a given feature or eliminate it altogether. As disclosed by Forman, the classifier (generated by training) can perform the feature scaling during a pre-processing step of the classification:

> Once classifier 18 has been generated, it can be used to classify unlabeled items 22. Each such item 22 is input into a pre-processing section 24 that preferably performs processing that is identical or at least substantially similar to that of pre-processing section 14 [(i.e., during training)]… Classifier 18 then operates on the modified feature set output from pre-processing section 24 to provide output predictions 27 of the underlying ground truth labels for the subject items 22… As indicated above, one aspect of the present invention is feature scaling prior to use of a feature set for training and/or classification purposes.
> (Ex. 1018, 3:56-4:11).

149. In Figure 1 of Forman, the pre-processing of both training (pre-processing section 14) and the pre-processing of classification (pre-processing 24) are shown in their respective stages. This figure is reproduced herein below:

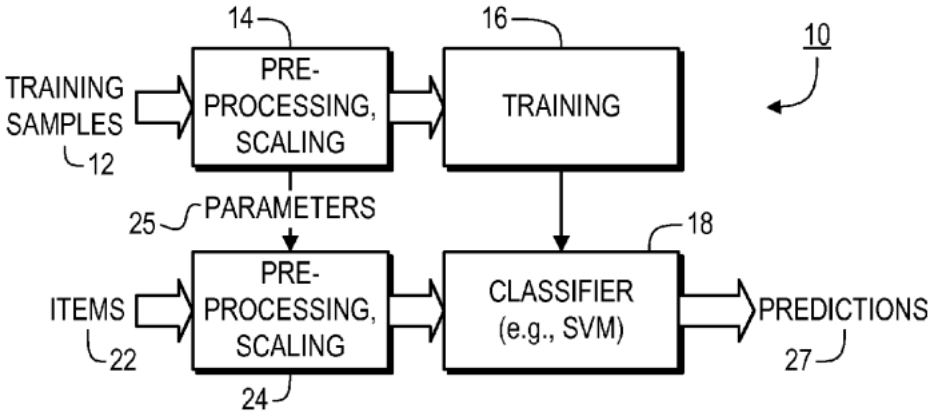


FIG. 1

150. Similarly, Figure 5 and Figure 6 show the steps of these two stages. These figures are reproduced herein below:

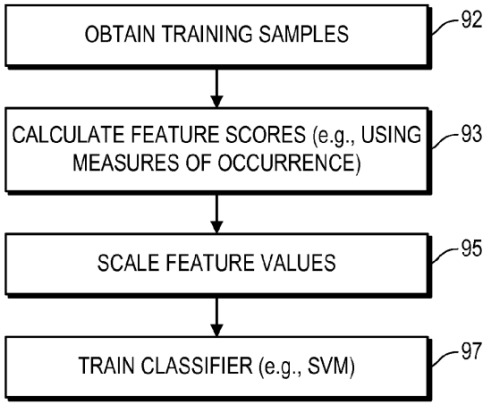


FIG. 5

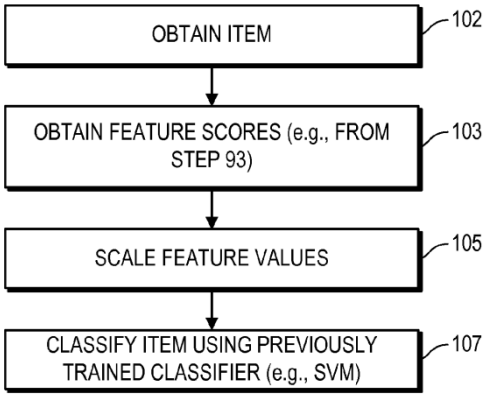


FIG. 6

151. Feature scaling uses scoring to know how to weight a feature (including eliminating it). One such method of scoring is called "Information Gain" and can be used especially when in conjunction with feature selection (e.g., during training).

> However, in the event feature selection is used in conjunction with the techniques of the present invention, and based solely on the limited experimentation performed to date, the currently preferred embodiments employ a different scoring technique for feature selection (e.g., Information Gain, as discussed in more detail below).
> (Ex. 1018, 3:24-31).

> In still further embodiments of the invention, entirely different scores instead are used. One example is Information Gain, which is defined as:
> IG=entropy(pos.neg)-(P(word)entropy(tp,fp)+(1-P(word))entropy(fn,tn))
> (Ex. 1018, 5:54-60).

152. The feature scaling can be used to turn features on or off (e.g., by "eliminating" a feature). According to Forman, the process is similar to ("mirrors") the same process used during training:

> Next, in step 93 a score is calculated for each of the training samples. In the preferred embodiments, the score comprises the BNS score referenced above. However, as also noted above, in other embodiments a different score is calculated. In certain sub-embodiments, all features are removed that have received a feature score that is zero (or very close to Zero based on a pre-defined threshold), because feature values multiplied by Zero are always zero and therefore cannot help discriminate positives from negatives.
> (Ex. 1018, 6:66-7:8).

> FIG. 6 is a flow diagram illustrating a process for classifying an item according to a representative embodiment of the present invention. ***As will become apparent below, several of the steps of the preferred process for classifying an item mirror those described above for training the classifier 18.***

> Initially, in step 102 an item 22 to be classified is obtained. As with the training samples 12, the item 22 preferably is represented by a set of values corresponding to a feature set 38, although in this case there generally is of course no pre-identified classification label. ***In certain embodiments, the feature set 38 has been modified from an original feature set, e.g., by eliminating certain features.***

(Ex. 1018, 7:56-67, emphasis added).

> In step 105, the feature values for the item 22 are scaled based on the scores that were obtained in step 93. This step, as well as the considerations pertaining to it, preferably are similar to step 95, discussed above. More preferably, the identical scaling that was performed in the training step 95 (used for the training of the classifier that will be used in step 107) is performed in this step 105.
> (Ex. 1018, 8:7-13).

153. Forman explains that different feature sets are better for different circumstances. Thus, one may have a larger set of features generally available and then turn some features on or off to reduce processing requirements in specific contexts. "In addition, in order to minimize processing requirements, the same feature set sometimes will be generated and used for multiple different purposes, so the resulting feature set has some features that are not particularly appropriate for the specific task at hand." (Ex. 1018, 2:67-3:5).

154. Forman explains that a probabilistic model can be used to weight the features: "in the preferred embodiments module 40 scores each feature using a score defined… [using] F^{-1} is the inverse standard Normal cumulative probability distribution function." (Ex. 1018, 4:43-62). Thus, not only can features be turned on or off but a probability distribution can be used to weight them ensuring that the output of a classification step is probabilistic as well. Figure 4 of Forman illustrates this function and is reproduced herein below:

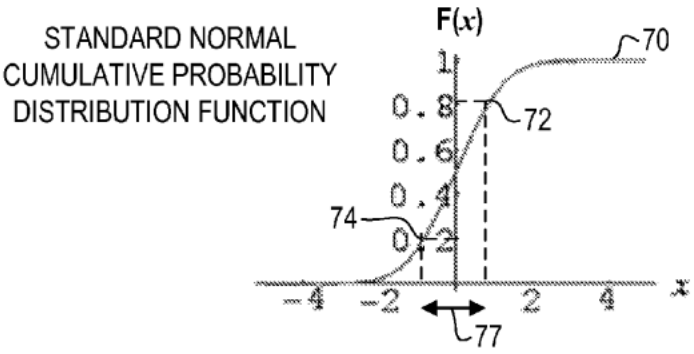


FIG. 4

(Ex. 1018, Figure 4).

155. In terms of application, Forman explains that the classifiers trained using the disclosed techniques can be used to identify spam. "The resulting [classifier] predictions 27 are then used, e.g., for any desired useful purpose… [such as] redirecting spam e-mail to a separate folder." (Ex. 1018, 4:4-6).

IX. GROUND 1: CLAIMS 1-4, 8, 10-25, 29, 32-39, 43, 46-53, AND 57 ARE RENDERED OBVIOUS BY TAHAN IN VIEW OF MENAHEM IN FURTHER VIEW OF FORMAN

156. As I describe in greater detail hereinbelow, a POSITA would have found Claims 1-4, 8, 10-25, 29, 32-39, 43, 46-53, and 57 rendered obvious by Tahan in view of Menahem in further view of Schultz.

A. Motivation to Combine Tahan, Menahem, and Forman

157. A POSITA would have been motivated to combine the teachings of Tahan, Menahem, and Forman.

158. As an initial, threshold matter, Tahan and Menahem are both 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. 1005, p. 1). 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. 1006, p. 1).

159. 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. 1005, p. 1).

160. 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. 1005, p. 11). Specifically, Tahan combines Mal-ID with three other classification technologies known as "Rotation Forest," "C4.5", and a "multilayer perception." (Ex. 1005, p. 13). Tahan further identifies that other classifiers were known in the art to be used for classifying malware, including Support Vector Machines (SVM). (*See* Ex. 1005, pp. 3-5: "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.). 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. 1005, p. 28).

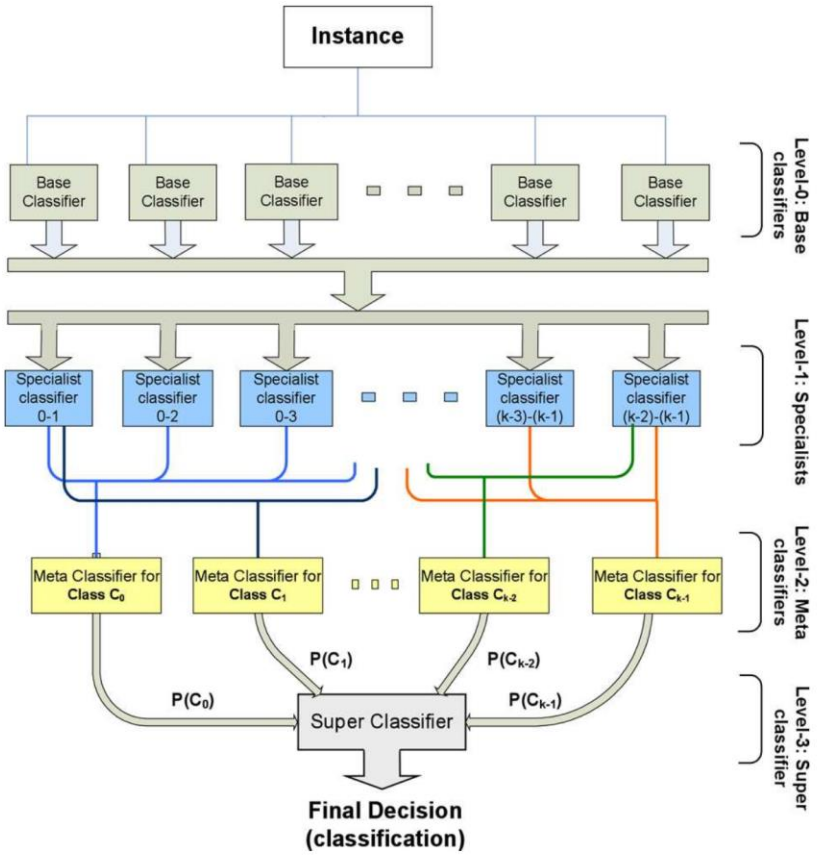
161. Because Tahan discloses that there is no method that is always superior, A POSITA would have been motivated to look for and try different techniques, classifier combinations, and feature combinations, in order to identify other solutions or methods for additional "strong points and suggested cases where [such methods] should be preferred over others" (and to achieve efficient and effective machine learning processes for detecting harmful executable files overall). (Ex. 1005, p. 28).

162. 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. 1006, p. 1). In addition, a POSITA would have looked to Menahem to enhance the meta-feature analysis of Tahan using these ensemble methods (e.g., using the entropy measurements of Tahan to select features to be analyzed by higher-level classifiers).

163. Menahem classifies different types of data (e.g., "n-grams, Portable Executable (PE) features, and Function-based features", Ex. 1006, p. 5) 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. 1006, p. 5). 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. 1006, p. 2: "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.).

164. 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. 1006, p. 11). 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. 1006, p. 4).

165. 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. 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. 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. 1006, p. 2). 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. 1006, p. 12: "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.).

166. A POSITA would have further been motivated to consider using 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).

167. A POSITA also would have been motivated to augment Tahan's teachings on the importance of features with Menahem's teachings on effective feature selection/reduction. As I explained in Section VIII.A. 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[f]; *see also* Ex. 1005, p. 3).

168. A POSITA would have recognized the importance of building on these teachings of Tahan. A POSITA would have looked to a reference like Menahem at least because Tahan cites Menahem as related research and also because Menahem 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. 1006, pp. 2 and 30).

169. 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. 1006, p. 12). 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. 1006, p. 2; *see id.*, p. 12; *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).

170. 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. 1006, p. 1).

171. 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.).

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

173. A POSITA would have been further motivated to combine the teaching of Tahan and Menahem with those of Forman. All three systems deal with machine learning classification techniques for identifying malicious data. Tahan and Menahem are directed toward recognizing binary malware while Forman addresses email-based malware (e.g., spam). (Ex. 1018, 2:39-58).

174. Moreover, and as discussed above, Tahan and Menahem also were aware of and recognized the wide-spread use of Support Vector Machines (SVM) which are used by Forman. (Ex. 1018, 7:30-55). Tahan and Menahem both suggested the use of many different types of machine learning techniques including SVM and Menahem explicitly experimented with using multiple methods combined into a single "ensemble." (Ex. 1006, p. 3). A POSITA would have been motivated to use the Support Vector Machines (SVM) techniques of Forman and incorporate them into their systems.

175. Both Tahan and Menahem recognized that there are too many features to be used in the machine-learning training process. Both references suggested reducing features during the training-phase. However, Forman expands on training-phase feature selection and introduces classification-phase feature selection using feature scaling. Forman explains that this enables a large feature set to be applied to a wider range of specific situations. A POSITA would have recognized that both Tahan and Menahem identify multiple types of binaries that may be classified.

176. For example, Tahan identifies malware that is obfuscated (which means the binary code is obscured and made difficult to reverse engineer) and non-obfuscated. As Tahan explains,

"In order to estimate the effect of obfuscation on detection rate, we have divided the tested malware into two groups—obfuscated and non-obfuscated… The results show that the TPR for both obfuscated and non-obfuscated group is very similar with slight advantage to detecting obfuscated malwares." (Ex. 1005, pp. 22-23). A POSITA would have been motivated to apply Forman's teachings to Tahan in order to narrow feature sets at classification time to be more relevant to these different types of binary malware.

177. Also of significance, Tahan also has classification-time feature scaling and selection. As described in Section VIII.A., Tahan discloses turning off certain features (within certain segments) when their entropy is out of range. 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. 1005, p. 9). 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. 1005, p. 10).

178. Tahan's disabling of entropy was not applied to Support Vector Machines. But a POSITA would have recognized that Forman's scaling works on similar principles. Forman, for example explains that features can be bounded to a certain range. Forman discloses: "if an item to be classified 22 has a feature value that fall outside the previously known range [Vmin, Vmax], then its value is clipped to fall within this range before proceeding with the scaling." (Ex. 1018, 8:13-17). A POSITA would have understood that because Forman also teaches eliminating features based on scaling that such features could be eliminated if they dropped below Vmin or rose above Vmax depending on the configuration.

179. Like Tahan, Menahem also identified different types of files. In particular, Menahem collected files that were binary executables as well as binary libraries (e.g., DLLs). In describing their collected dataset, Menahem explains, "[w]e collected a large set of 22,735 benign files and 7690 malware that were identified as Win32 executable and DLLs." (Ex. 1006, p. 6). A POSITA would have been motivated to apply Forman's teachings to Menahem in order to narrow feature sets at classification time to be more relevant to these different types of binary files.

180. Notably, Menahem identified the very problem that Forman aims to solve: processing time. In Menahem, the choice was made to reduce the side of the dataset: "due to processing power and memory limitations we could not run our ensemble experiments on the entire datasets and we had to reduce the datasets in order to complete the experiments in reasonable time." (Ex. 1006, p. 7). A POSITA would have been motivated to apply Forman's teachings of dynamically reducing features in order to test on a larger dataset.

181. Menahem also identified working with different datasets on different component classifiers as a future goal. In describing future work, Menahem states, "In future experiments we would like to evaluate the performance of inducers that are trained on different features." (Ex. 1006, p. 13). A POSITA would have understood that the current model of having classifiers (inducers) trained on the same dataset could be adapted to use a different dynamic set at classification time as taught by Forman. (Ex. 1018, 7:30-45). This would achieve similar results but without require retraining base classifiers.

182. I also note that Tahan and Menahem identify the "Information-Gain" feature selection technique as one used in the art. (Ex. 1005, p. 3; Ex. 1006, p. 3). Forman also uses Information-Gain for feature selection and feature scaling in the classification phase.

183. In summary, a POSITA would have found Forman's teachings compatible and a natural fit for Tahan and Menahem. Moreover, a POSITA would have been motivated to obtain Forman's benefits for reduced computation and better applicability of feature sets to the Tahan and Menahem combination.

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

184. Each of the limitations of Claim 1 of the '869 patent is rendered obvious by Tahan in view of Menahem in further view of Forman as described below.

1. Limitation 1[a]

185. Limitation 1[a] of the '869 patent reads:

A computer-implemented method comprising:

186. 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 '869 patent is considered limiting, Tahan discloses the preamble of Claim 1.

187. 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. 1005, p. 8).

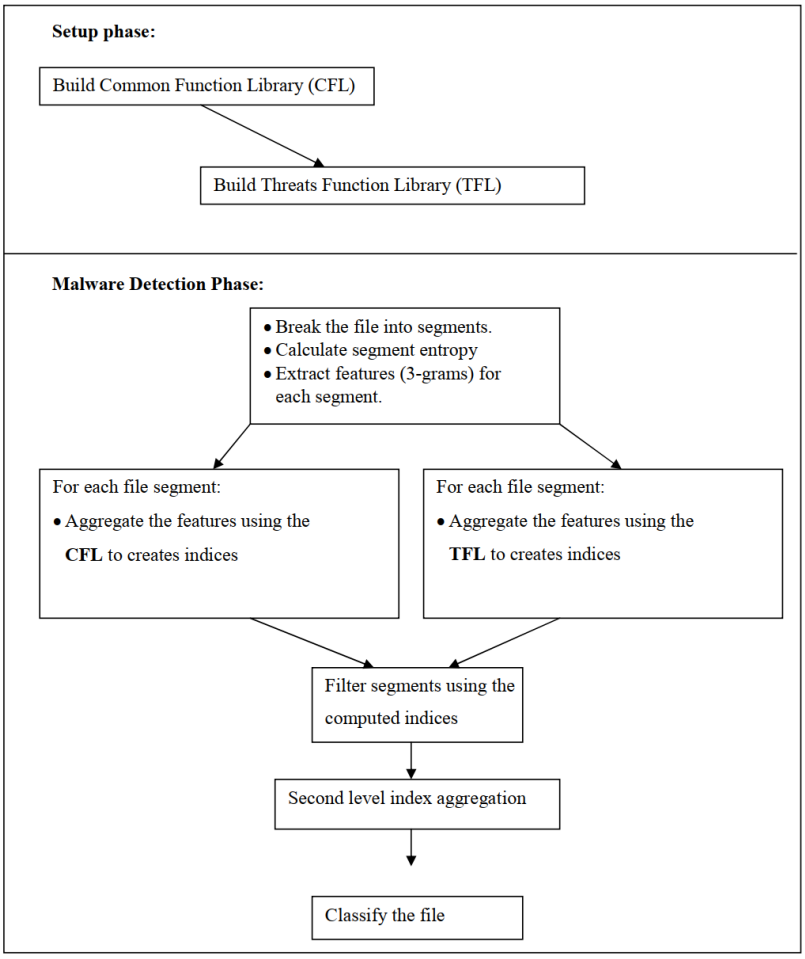


Figure 1: The Mal-ID method for detecting new malware applications.

188. 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." (Ex. 1005, p. 1, emphasis added).

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

2. Limitation 1[b]

190. Limitation 1[b] of the '869 patent reads:

identifying, by a knowledge module, static data points that may be indicative of either a harmful or benign executable file;

191. Tahan discloses limitation 1[b] of the '869 patent.

192. The ’869 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.” (Ex. 1001, 5:61-67).

193. I note here that while Claim 1 of the ’869 patent uses the term “knowledge module,” that term does not appear in the ’869 patent specification. Instead, the term “knowledge component 102” is used throughout the specification to describe the claimed functionality. As I outlined in Section VI.A., the “knowledge component 102” comprises several operations, including collecting, parsing, storing, and managing training data used for building and training/re-training a classifier, as well as extracting data from executable files to be evaluated by the classifier. The component also includes (or is) storage for this data. (*See* Section VI.A.; *see also* Ex. 1001, 3:31-8:34).

194. In particular, the knowledge component of the ’869 patent collects and maintains a collection of data about benign and malicious files:

> Knowledge component 102 described herein may collect data for use in building, training and/or re-training a learning classifier to evaluate executable files. The knowledge component 102 is one or more storages, memories, and/or modules that continuously collects and manages data that may be used to detect threats in files such as executable files. In one example, the knowledge component 102 maintains a robust collection of data comprising known malicious executable files, known benign executable files, and potentially unwanted executable files.
> (Ex. 1001, 3:54-63).

195. Furthermore, the knowledge component processes static data points that may be indicative of malware:

> In examples, the collected data on the executable files is analyzed to identify static data points that may indicate one of a malicious file, a

> benign file or a potentially unwanted file. For instance, the knowledge component 102 may employ one or more programming operations to identify static data points for collection, and to associate the static data points with one of the categories of files (e.g., malicious, benign or potentially unwanted). Programming operations utilized by the knowledge component 102 include operations to collect file data (e.g., executable files or data points from executable files), parse the file data, and store extracted data points. In at least one example, the knowledge component 102 comprises one or more components to manage file data. For example, the knowledge component 102 may comprise one or more storages such as databases, and one or more additional components (e.g., processors executing programs, applications, application programming interfaces (APIs), etc.).
> (Ex. 1001, 5:8-52).

196. Tahan discloses a process or "algorithm" (e.g., labeled "Setup phase" in Figure 1, reproduced hereinabove) for building two "repositories" of file segments (3-grams) by examining the segments and selecting relevant ones. (Ex. 1005, pp. 8-9). These repositories are called the Common Function Library (CFL) and the Threat Function Library (TFL). "The CFL contains data structures constructed from benign files," and "[t]he TFL contains data structures constructed from malware". (Ex. 1005, p. 7). The data structures from these executable files are, at the very least, "program information," an example of static data points described in the '869 patent. (Ex. 1005, p. 4).

197. The known-benign and known-malware files are retrieved from trusted sources and then broken down into segments to create the CFL and TFL:

> The setup phase involves collecting two kinds of files: benign and malware files. The benign files can be gathered, for example, from installed programs, such as programs located under Windows XP program files folders. The malware files can, for example, be downloaded from trusted dedicated Internet sites, or by collaborating with an anti-virus company. In this study the malware collection was obtained from trusted sources. 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. 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 to either benign (CFL) or malware (TFL). Moreover, 3-grams that seem to appear approximately within the same offset in all malware can be used to characterize the malware. Before calculating the 3-grams, the training files are randomly divided into 64 groups.
> (Ex. 1005, p. 7).

198. A POSITA would have recognized the process or algorithm that creates and utilizes such repositories, or knowledge base, corresponds to the "knowledge component 102" taught in the '869 patent specification:

> The data collected and maintained by the knowledge component 102 yields a knowledgebase that may be used periodically to train a classifier, e.g. learning classifier component 104 utilized by system 100.
> (Ex. 1001, 4:39-43).

199. Thus, Tahan discloses limitation 1[b].

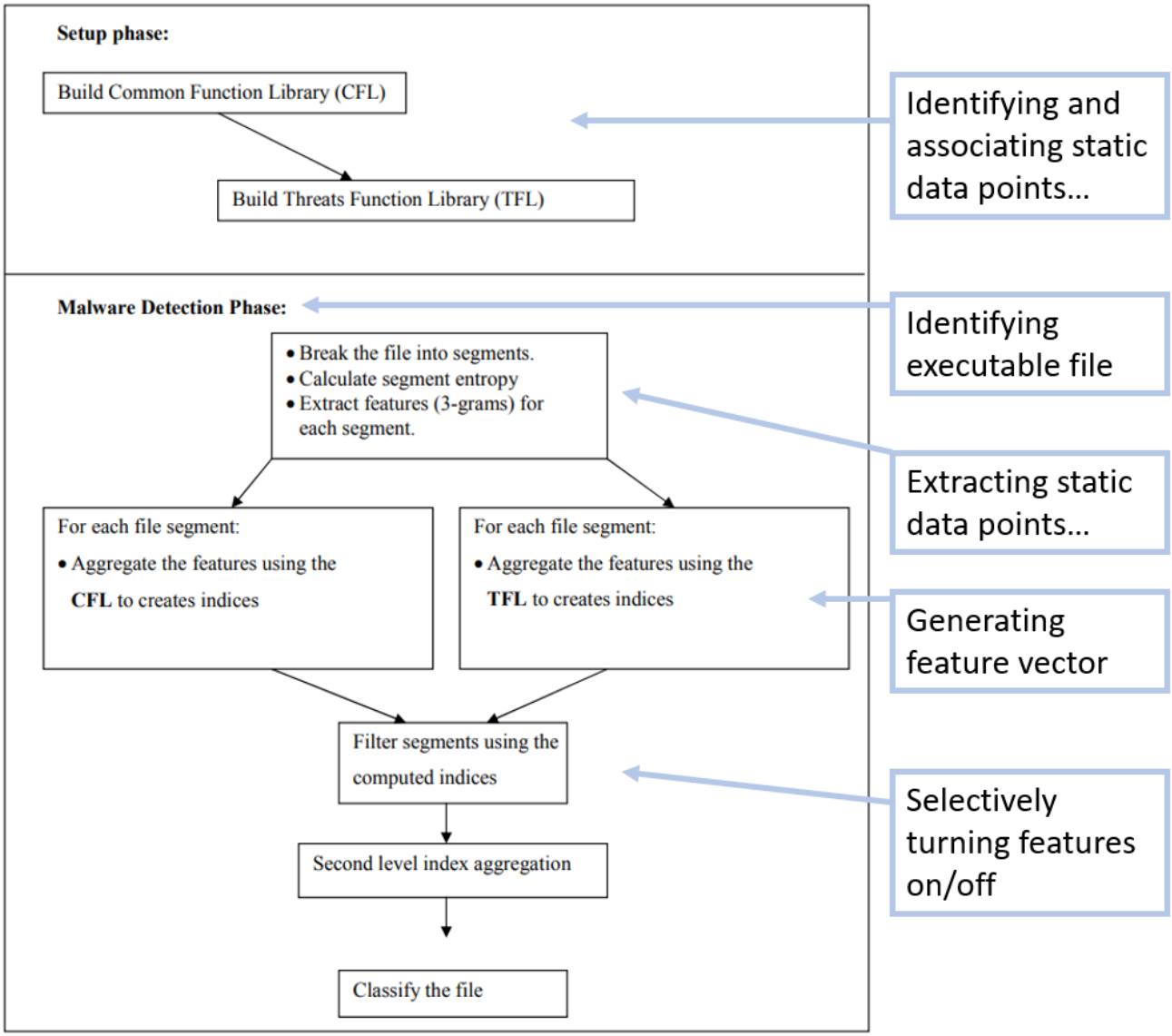
3. Limitation 1[c]

200. Limitation 1[c] of the '869 patent reads:

> *associating, by the knowledge module, the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;*

201. Tahan discloses limitation 1[c] of the '869 patent.

202. As I discussed with reference to limitation 1[b], Tahan discloses identifying static data points with harmful files and benign files. These aspects are illustrated in relation to Figure 1 of Tahan below in the annotated figure:



203. Furthermore, Tahan teaches the creation of a Common Function Library and a Threat Function Library, and associates particular data points either with benign files or with harmful files:

> 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 to either benign (CFL) or malware (TFL).*** Moreover, 3-grams that seem to appear approximately within the same offset in all malware can be used to characterize the malware. Before calculating the 3-grams, the training files are randomly divided into 64 groups.
> (Ex. 1005, p. 7).

204. Thus, Tahan discloses that the repository building algorithm calculates some metadata (3-grams) and examines it before associating the segment with a benign library or a threat library. A POSITA would have recognized the algorithm that creates and utilizes such repositories to correspond to the "knowledge component 102" taught in the '869 patent specification:

> In examples, the collected data on the executable files is analyzed to identify static data points that may indicate one of a malicious file, a benign file or a potentially unwanted file. For instance, the knowledge component 102 may employ one or more programming operations to identify static data points for collection, and to associate the static data points with one of the categories of files (e.g., malicious, benign or potentially unwanted). Programming operations utilized by the knowledge component 102 include operations to collect file data (e.g., executable files or data points from executable files), parse the file data, and store extracted data points. In at least one example, the knowledge component 102 comprises one or more components to manage file data. For example, the knowledge component 102 may comprise one or more storages such as databases, and one or more additional components (e.g., processors executing programs, applications, application programming interfaces (APIs), etc.).
> (Ex. 1001, 5:8-52).

205. Thus, Tahan discloses limitation 1[c].

4. Limitation 1[d]

206. Limitation 1[d] of the '869 patent reads:

identifying an executable file to be evaluated;

207. Tahan discloses limitation 1[d] of the '869 patent.

208. Tahan discloses "identifying an executable file to be evaluated." Tahan discloses a "setup phase" for creating a classifier that identifies executable files to be evaluated as "benign" or "malware." (Ex. 1005, p. 7). Tahan provides one example of files evaluated as "malware," received by being "downloaded from a trusted Internet site, or by collaborating with an anti-virus company." (*Id.*). On the other hand, Tahan explains that an executable file taken from the

Windows operating system would be "benign." (*Id.*). As described later in Tahan, new malware applications are detected in much the same way. The Mal-ID algorithm analyzes a new application by inputting files, which a POSITA would understand to be executable files, and breaking those files into segments for further analysis and ultimately classification of the file or application. (Ex. 1005, pp. 8-9).

209. Tahan's disclosures are similar to what is described in the '869 specification:

> For example, ***a research team may be employed to continuously collect new examples of harmful executables, benign executables, and potentially unwanted executables***, as many unknown executable files are generated on a daily basis over the Internet. ***Executable files may be evaluated by the research team*** such as by applying applications or processing to evaluate executable files including data associated with the file and/or actions associated with the file (e.g., how it is installed and what a file does upon execution). Continuous update of the data maintained by the knowledge component 201 in conjunction with on-going re-learning/re-training of the learning classifier component 104 ensures that system 100 is up-to-date on current threats.
> (Ex. 1001, 4:54-67, emphasis added).

210. Thus, Tahan discloses limitation 1[d].

5. Limitation 1[e]

211. Limitation 1[e] of the '869 patent reads:

extracting, by the knowledge module, a plurality of static data points from the identified executable file;

212. Tahan and Menahem each disclose limitation 1[e] of the '869 patent.

213. The '869 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." (Ex. 1001, 5:61-67).

214. Table 1.1 of the '869 patent, reproduced hereinbelow, provides a few examples 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 version	language	original file	Entry Point Anomaly
file version number	file flags masks	private build	certificate Validity
product version number	file flags	product name	Certificate Exception
size of heapr	file OS	special build	Code Characteristics Anomaly
size of stackr	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 Subject	Number of Functions 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 section strings	PE Magic Validity
	resource code page		Resource Exception
	resource size		VR Code Ratio Anomaly
	DLL characteristics		Import Exception

(Ex. 1001, Table 1.1).

215. As I discussed in Section VI.A., the '869 patent discloses that the "knowledge component 102" has a broad scope of functionality and is used not just for collecting and managing training data but also for extracting static data points from executable files to be classified. (Ex. 1001, 7:1-5; *see also* Section VI.A.).

216. Tahan discloses "extracting" features (from executable files) that include examples of "static data points" identified by the '869 patent specification. As I explained with respect to

the background topic on machine learning techniques, features ("static data points") are extracted for both training the classifier and for classifying unknown samples. (*See* Section V.A.).

217. In explaining the state of the art, 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. 1005, pp. 2-3, 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. 9, emphasis added). I discuss these extracted features ("static data points") below.

218. Tahan discloses that the feature extraction for detection of malware in unknown or new files and applications follows the same process as in building the CFL/TFL repositories. In both cases, the files are segmented into 3-grams, then evaluated with meta-data, and then filtered before assignment to a classification or category (e.g., malware or TFL). (Ex. 1005, pp. 7-10). A POSITA would have understood that code or modules used to implement the CFL/TFL repository building algorithm are also useful for extracting static data points for classification, as in this limitation. For example, Tahan Figure 2 contains pseudocode for the "detection phase" that includes functions named "GenerateSegmentCollection" (which subdivides an input file into segments) and "SegmentTo3Grams" (which subdivides a segment into 3-grams), two operations

that are necessary both for building the CFL/TFL libraries and for extracting data from an executable file to be classified. (Ex. 1005, p. 12). A POSITA would have been motivated to write a shared module to perform these operations for both phases, in order to gain efficiency while building the system as well as to simplify maintenance of the system during its lifespan. A shared module such as this fits within the scope of the claimed "knowledge module". (*See* Section VI.A. about the broad range of functions performed by the '869 patent's "knowledge module"). A POSITA would have understood that the process for extracting features from known files during training should be as close as possible to the actual extraction of features during detection and classification so that the classifier is appropriately trained and so that extracted data used for classification conforms to the format and content expected by the classifier. Figure 2, containing pseudocode of this process, is reproduced hereinbelow:

```
1   SegmentColl=GenerateSegmentCollection(FileContent,SegmentsRequired,SegmentLen);
2   SegmentCheck=0;
3   ForEach Segment in SegmentColl do
4       {
5       Entropy  = Entropy(Segment.string);
6       If (Entropy<=EntropyLow) or (Entropy>= EntropyHigh) then
7          {SegmentColl.delete(Segment); continue; }
8
9       Segment3Grams:=SegmentTo3Grams(Segment);
10
11      CFL_MFG = CFL.Count_Files_With_All_3gram (Segment3Grams)
12      If (CFL_MFG>0) then { SegmentColl.delete(Segment); continue; }
13
14      TFL_MFG = TFL.Count_Files_With_All_3gram (Segment3Grams)
15      If (TFL_MFG< ThreatsThreshold) then { SegmentColl.delete(Segment); continue; }
16
17      TFL_spread   = TFL.CalcSpread (Segment3Grams);
18      If (TFL_spread =0) or  (TFL_spread >SR)  then
19         {SegmentColl.delete(Segment); continue; }
20
21      // optional stage
22      SegmentCheck++;
23      If (SegmentCheck>log(SegmentLen)*NumberOfMalwareInTraining) then continue;
24      InMalwareFile  = TFL.SearchInMalwareFiles(Segment);  //search by bit-fields
25      If  not InMalwareFile  then { SegmentColl.delete(Segment); continue; }
26      }
27
28  SegmentsInMalwareOnly = 0;
29  ForEach Segment in SegmentColl do
30      { SegmentsInMalwareOnly  = SegmentsInMalwareOnly +1; }
31
32  Malware_Classfication_Result = SegmentsInMalwareOnly > ThreatSegmentThreshold;
```

(Ex. 1005, Figure 2).

219. Tahan discloses 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. 1005, p. 3). Furthermore, "In this paper, the term segment refers to a sequence of bytes of certain size that was extracted from an executable file." (Ex. 1005, p. 2). 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. 1005, p. 4, emphasis added).

220. 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.). Tahan describes the extraction of features for analysis in malware detection as involving similar segmentation and categorization as in the setup phase. For example, the files are broken into segments and those segments are categorized into CFL segments or TFL segments based on meta-data as shown in Figure 1. (Ex. 1005, p. 8-9).

221. 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. 1005, p. 2)

222. A POSITA would have recognized that Tahan describes many "[f]eatures extracted from PE executables" that the '869 patent specification also teaches. For example, the following table illustrates some parallels a POSITA would have recognized:

Tahan (Ex. 1005, p. 2)	**'869 patent specification at 5:61-67**	**'869 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	string and flag information resource information	• resources • product version • private build • special build

kinds of information. For example, strings for dialogs, images, dialog structures; version information, build information, original filename, etc.)		• internal name

223. Menahem also discloses "extracting, by the knowledge module, a plurality of static data points from the identified 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. 1006, p. 2, emphasis added; *see also id.*, p. 12; 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":

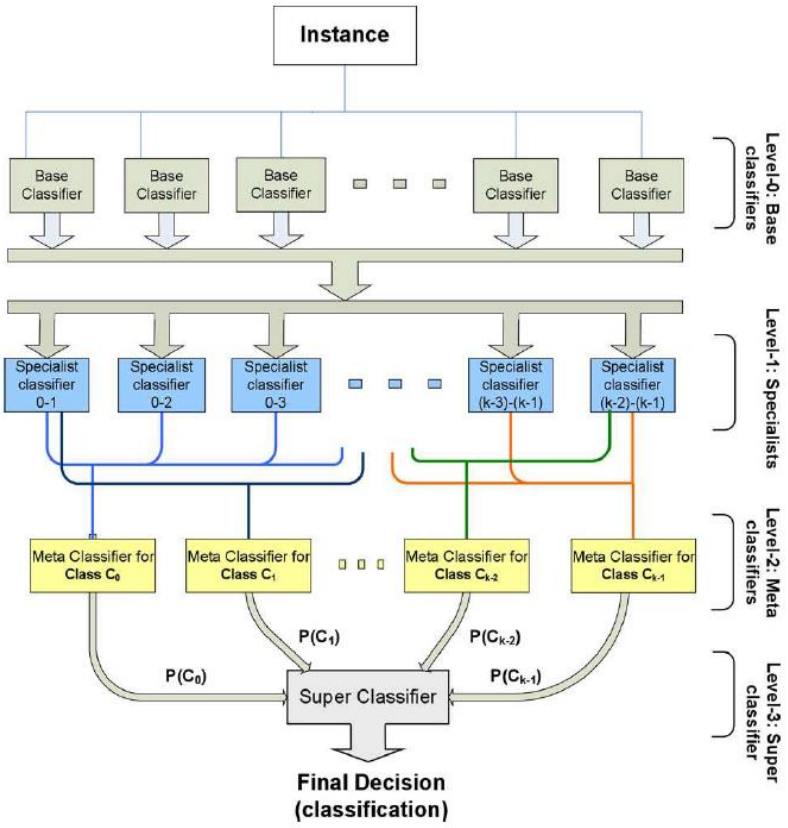


Fig. 1. Troika Architecture reveals three layers of combining classifiers; the Specialist classifiers combine the outputs of the base-classifiers, Meta-classifiers combine the Specialist classifier outputs, and finally, the Super-classifier, combines all Meta-classifier outputs. The Super-classifier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

(Ex. 1006, Fig. 1.)

224. Menahem explicitly discloses extracting (from executable files) similar types of data to those extracted by Tahan, including 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. 1006, p. 5, emphasis added). All three types of features disclose the claimed "static data points" for the reasons discussed above and those discussed below.

225. 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. 1006, p. 12).

226. 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. 1006, pp. 5-6).

227. 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. 1006, p. 6). 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. 1006, p. 6).

228. Thus, Tahan and Menahem each disclose the requirements of limitation 1[e]. To the extent that a combination is needed or used in subsequent limitations, a POSITA would have been motivated to combine Tahan, Menahem, and Forman, as I explained in Section IX.A. For example, 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. 1006, p. 2). Substituting an ensemble classifier for a single base classifier was known in the art, as discussed in prior art and in the '869 patent itself. (*See*, e.g., Ex. 1001, 9:27-54; *see also* my explanations in Section VIII.B.).

229. 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. 1006, p. 2). 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. 1006, p. 12; *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. 1006, p. 2; *see also id.*, p. 12; *see also* my analyses and explanations in Section VIII.B.).

230. 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. 1006, p. 1). In doing so, a POSITA would have taken advantage 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.

231. Thus, Tahan and Menahem each disclose limitation 1[e].

6. Limitation 1[f]

232. Limitation 1[f] of the '869 patent reads:

> *generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are*

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

selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

233. Tahan in view of Menahem in further view of Forman discloses limitation 1[f] of the '869 patent. I have decomposed this limitation into three elements.

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

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

235. 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[e]), 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. 1005, pp. 2-3, emphasis added).

236. 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."

237. Tahan also discloses "generating a feature vector from the plurality of static data points using a classifier trained to classify the 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. 1005, p. 1 ("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[g].)

238. 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 '869 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 I 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. 1009, 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.*, 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 '869 patent. (*See* Section VI.A.).

239. The '869 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." (Ex. 1001, 12:55-60, 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.

240. Thus, a POSITA would have understood that machine learning mechanisms/modules/applications in Tahan disclose classifiers that generate the feature vector as required by this claim limitation. (*See also* my analysis for limitation 1[g].) 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. 1005). For example, as discussed in Sections VIII.A and IX.A., 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."

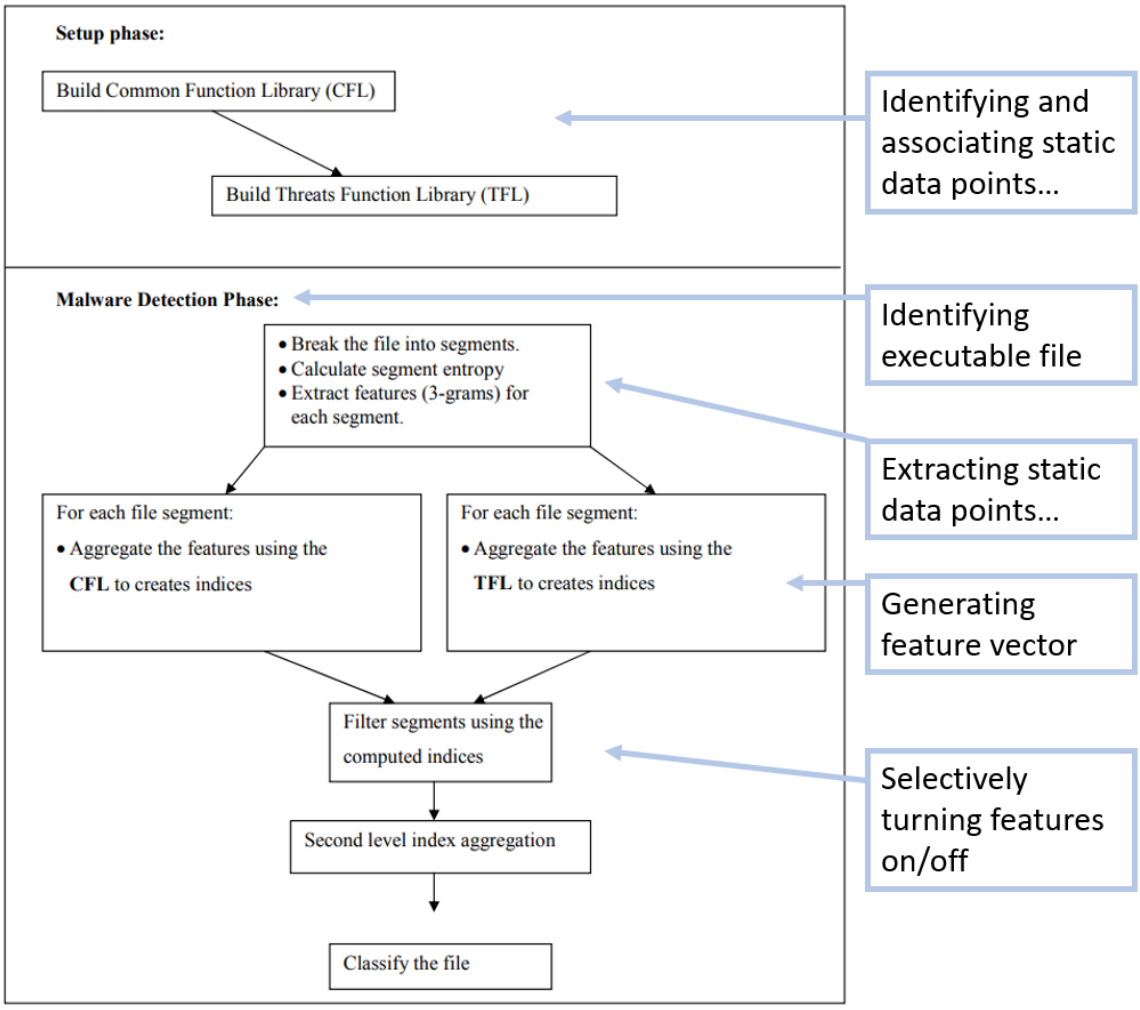
241. Moreover, Tahan also discloses that such a classifier is "trained to classify the static data points." Moreover, Tahan discloses that such a classifier is "trained to classify the static data points." For example, as I discussed above and for Section VIII.A., Tahan's classifiers (such as Mal-ID) 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 static data points" (e.g., as malicious or benign)). (I incorporate by reference my analysis for Section VIII.A. and limitation 1[b].)

242. 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. 1005, p. 4).

243. 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. 1005, p. 7). 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.*).

244. 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. 1005, p. 10). On the other hand, "segments that actually reside in the malware are left in the segment collection." (Ex. 1005, p. 11).

245. 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 a threat 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. 1005, p. 7 ("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).").)



246. 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 static data points."

247. Also, Tahan and Menahem disclose "a classifier trained to classify the static data points" (e.g., classify static data points into categories or data types) for the reasons discussed for Claim 2. I incorporate by reference my analyses for Claim 2.

248. 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. 1005, p. 1; *see also, e.g.,* Ex. 1005, pp. 4, 6, 7, 9-10).

249. The '869 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).
> (Ex. 1001, 7:32-42).

250. 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 '869 patent of "a classifier trained to classify". Tahan's description of the approach teaches feature selection and the creation of a "feature vector". As I explained with reference to limitation 1[e], 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 static data points".

251. 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. 1006, pp. 5-6).

252. 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. 1006, p. 12). 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, including SVMs. (Ex. 1006, p. 2, emphasis added; *see also id.*, p. 12; *see also* my analyses and explanations in Sections V.A. and VIII.B.). Thus, a POSITA would have understood Menahem's super classifiers or ensemble classifiers are trained to classify static data points based on training data that generate feature vectors as required by the claim language.

253. Forman also teaches generating a feature vector using a learning classifier component. Forman explains that text classifying, similar to the text classifying described by both Tahan and Menahem, can be represented with values that go into a feature vector for classification. Forman explains that these concepts are well known in the "the conventional literature:"

> For example, for purposes of classifying an input e-mail message to determine whether the message is spam or non-spam, t***he feature set in one embodiment of the invention pertains to a list of words, with each the data field for each feature (i.e., word in this example) intended to hold a binary value indicating whether the word is present in the e-mail message***. ***In alternate embodiments, the feature set accommodates entry of an integer number for each word, indicating the number of occurrences of the word in the e-mail message***. Various techniques for selecting a feature set, with the particular technique typically depending upon the particular classification problem, are discussed in the conventional literature.

(Ex. 1018, 2:47-58, emphasis added).

254. Although Tahan discloses this element, a POSITA would have also looked to Menahem and Forman for additional details, such as Menahem's ensemble classifier for better classification results and Forman's disclosures about how to encode textual data in a feature vector. I have already explained in Section IX.A. why a POSITA would have been motivated to combine Tahan, Menahem and Forman and I incorporate this analysis by reference.

255. Thus, Tahan alone or in combination with Menahem and Forman discloses this element.

"based on training data, the training data comprising files known to fit into one of a plurality of categories of files,"

256. This is the second element, disclosed by Tahan alone or in combination with Menahem and Forman. 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. 1005, p. 7). Tahan includes, in its categorization of malware, "spyware", which a POSITA would have understood is known unwanted software (including unwanted executable files). (Ex. 1005, pp. 1-2). 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. 2, emphasis added). This is consistent with the '869 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***." (Ex.

1001, 4:7-11, 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.*)

257. 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. 1005, p. 7). 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. 1005, p. 13, emphasis added).

258. A POSITA would have understood that Tahan teaches the claimed "training data comprising files known to fit into one of the plurality categories of files," (described in limitation 1[c] as "including harmful files and benign files"), and that this data set is used to train the claimed "classifier trained to classify the static data points", described in the preceding element.

259. 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. 1006, p. 5, emphasis added). Menahem further discloses using this data set to train classifiers (which it calls "inducers"[4]): "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. 1006, p. 5, 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. 1006, p. 6).

260. A POSITA would have recognized that both Tahan and Menahem teach training a classifier (the claimed "classifier trained to classify the static data points") using a data set, and that such data sets represent the same concept as the claimed "training data comprising files known to fit into one of the plurality categories of files".

261. Tahan teaches a way of narrowing the scope to the most useful functions (e.g., functions in the TFL are more likely to be predictive of malware than functions in the CFL), thus improving the efficiency of the processing. (*See, e.g.,* Ex. 1005, p. 7, "disregarding" CFL segments in potential malware to reduce the quantity of segments that must be evaluated; *see also* my analyses of limitations 1[b] and 1[c]). This is done by creating two libraries, one containing functions common to many executables and the other containing functions seen only in malware files:

> To implement the suggested approach, two kinds of repositories are defined:
>
> 1. CFL—Common Function Library. The CFL contains data structures constructed from benign files.

[4] As I explained in Section V.A., the technique used to create (or train) a classifier is sometimes called an "inducer."

> 2. TFL—Threat Function Library. The TFL contains data structures constructed from malware without segments identified as benign (i.e., segments that appears in benign files).
> (Ex. 1005, p. 7).

262. A POSITA would have been motivated to apply Tahan's teachings to make Menahem's technique more effective, because while Menahem acknowledges the usefulness of features based on functions, Tahan describes how to narrow their scope. Tahan describes the motivation as follows:

> As noted above, many applications and malware are developed using the development platforms that include large program language libraries. The result is that large portions of executable code originate from the program language libraries. For example, a worm malware that distributes itself via email may contain a benign code for sending emails. ***Consequently, since the email handling code is not malicious and can be found in many legitimate applications, it might be a good idea to identify code portions that originate from a benign source and disregard them when classifying an executable file.*** 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.
> (Ex. 1005, p. 7, emphasis added).

263. To the extent that Tahan does not disclose using these features (e.g., the CFL and TFL) to train and classify with support vector machines, Menahem provides motivation for using similar "Function-based features" with various machine learning techniques, including support vector machines. (*See* Ex. 1006, pp. 2, 5). Moreover, Forman explicitly uses Support Vector Machines to categorize into either binary or multiple classes, for example by using non-linear SVMs. (Ex. 1018, 7:39-45). As Forman explains, "In the preferred embodiments of the invention, training section 16 implements Support Vector Machine (SVM) training in order to output its classifier 18." (Ex. 1018, 3:46-49). Additionally, Forman notes that binary classifiers, like SVM,

are often extended to multi-classification (classifying into more than two labels) using multiple classifiers. A single classification of the many (1-of-n) can be determined "by decomposing the 1-of-n problem, pitting each class against the others." (Ex. 1018, 2:23-24). Alternatively, it is possible to identify many matching classes out of the set of total classes (n-of-m) "by applying m independent binary classifiers to each item." (Ex. 1018, 2:26).

264. I have already explained in Section IX.A. why a POSITA generally would have been motivated to combine Tahan and Menahem and Forman, and I incorporate this analysis by reference.

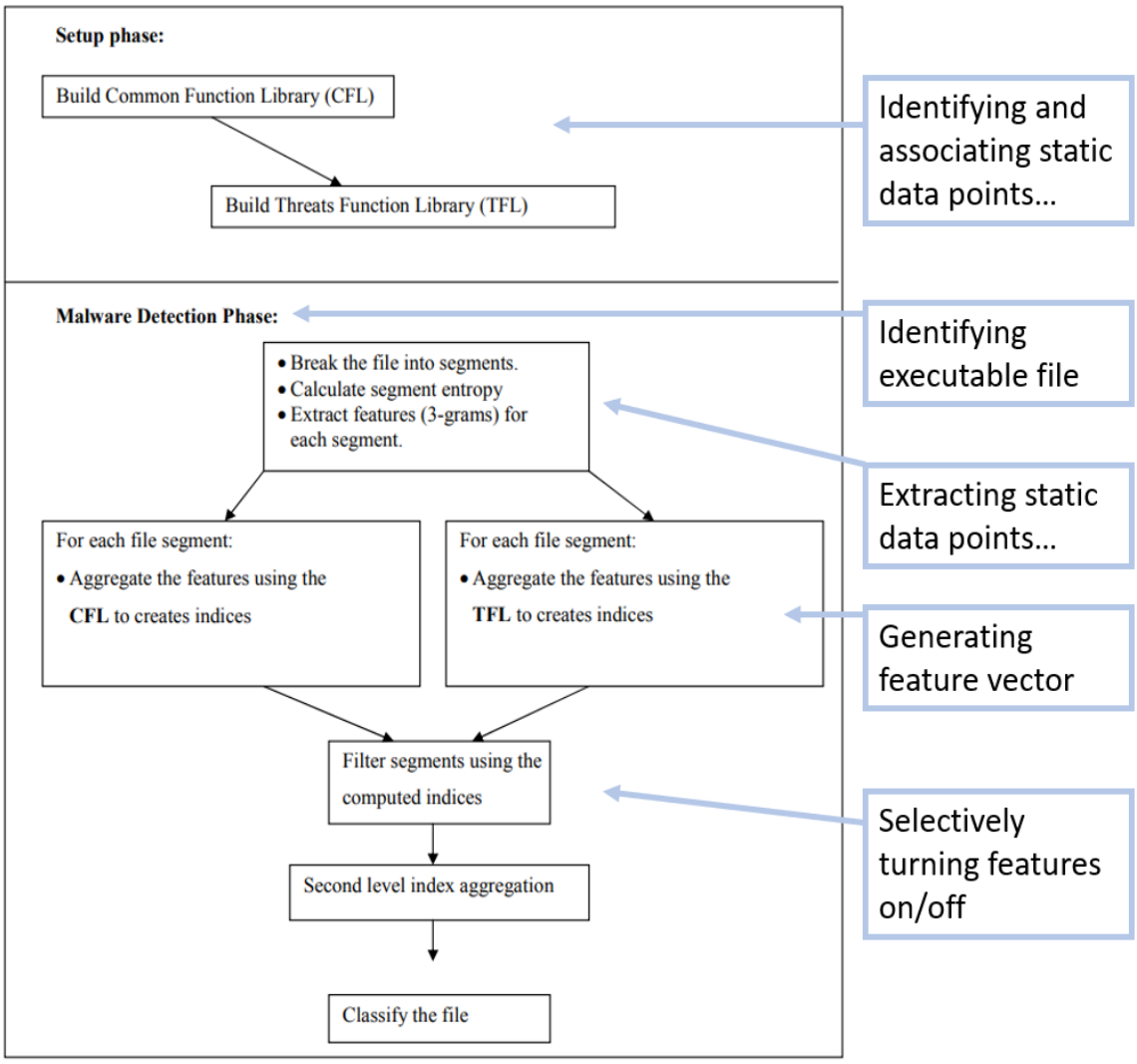
265. Thus, Tahan alone or in combination with Menahem and Forman discloses this element.

"wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range;"

266. This is the third element. Tahan in view of Menahem in further view of Forman disclose this element. Tahan and Menahem each emphasize that turning on or off features is an essential component of the training process. Tahan also teaches turning features on or off during classification. Finally, Forman teaches turning on or off features during the classification process and explains that techniques for selecting features during training can be applied during classification.

267. Tahan discloses turning on or off features *during classification* and "based at least in part on evaluation of whether a value of one of the plurality of 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. 1005, p. 10, emphasis added). 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. 1005, p. 10, emphasis added; *see also* my analyses and explanations for Sections VIII.A and limitation 1[e]). 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[f]). Thus, a POSITA would have understood that the segments are part of feature vectors (e.g., features of feature vectors). (*See id.*). See also Figure 1 of Tahan annotated below.



(Ex. 1005, Figure 1, annotated).

268. A POSITA would have understood that this technique teaches 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" and specifically during the classification process.

269. Moreover, as I discussed above for limitation 1[e], entropy is a static data point (which is consistent with disclosures of the '869 patent). (*See* my analysis for limitation 1[e]).

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.

270. In another example, Tahan discloses turning features on or off during the classification process by disregarding segments of the file to be classified that are present in the Common Function Library (CFL, the repository of segments found in benign files). Tahan explains:

> Once the setup phase has constructed the CFL and the TFL, it is possible to classify a file F as benign or as malware using the algorithm presented in Figure 2.
>
> 1. Line 1. Divide file F into S segments of length L. All segments are inserted into a collection and any duplicated segments are removed. The end result is a collection of unique segments. The Mal-ID implementation uses 2000 segments that are 64-bytes in length.
>
> 2. Line 3. For each segment in the collection:
>
> (a) Line 5. Calculate the entropy for the bytes within the segment.
>
> (b) Line 6. The algorithm gets two parameters EntropyLow and EntropyHigh. The entropy thresholds are set to disregard compressed areas (such as embedded JPEG images) and long repeating sequences that contain relatively little information. In this line we check if the entropy is smaller than EntropyLow threshold or entropy is larger than EntropyHigh. Is so then discard the segment and continue segment iteration. Preliminary evaluation has found the values of EntropyLow=0.5 and EntropyHigh=0.675 maximize the number of irrelevant segments that can be removed.
>
> (c) Line 9. Extract all 3-grams using 1 byte shifts.
>
> (d) Line 11. Using the CFL, calculate the CFL-MFG index.
>
> (e) Line 12. ***If the CFL-MFG index is larger than zero, then discard the segment and continue segment iteration. The segment is disregarded since it may appear in benign files.***
>
> (Ex. 1005, p. 10, emphasis added).

271. A POSITA would have understood disregarding a segment at classification time based on the calculated "CFL-MFG index" to teach the concept of features being "selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of

static data points is within a predetermined range". Figure 2, containing pseudocode of this process, is reproduced hereinbelow:

```
1   SegmentColl=GenerateSegmentCollection(FileContent,SegmentsRequired,SegmentLen);
2   SegmentCheck=0;
3   ForEach Segment in SegmentColl do
4       {
5       Entropy  = Entropy(Segment.string);
6       If (Entropy<=EntropyLow) or (Entropy>= EntropyHigh) then
7          {SegmentColl.delete(Segment); continue; }
8
9       Segment3Grams:=SegmentTo3Grams(Segment);
10
11      CFL_MFG = CFL.Count_Files_With_All_3gram (Segment3Grams)
12      If (CFL_MFG>0) then { SegmentColl.delete(Segment); continue; }
13
14      TFL_MFG = TFL.Count_Files_With_All_3gram (Segment3Grams)
15      If (TFL_MFG< ThreatsThreshold) then { SegmentColl.delete(Segment); continue; }
16
17      TFL_spread   = TFL.CalcSpread (Segment3Grams);
18      If (TFL_spread =0) or  (TFL_spread >SR)  then
19         {SegmentColl.delete(Segment); continue; }
20
21      // optional stage
22      SegmentCheck++;
23      If (SegmentCheck>log(SegmentLen)*NumberOfMalwareInTraining) then continue;
24      InMalwareFile  = TFL.SearchInMalwareFiles(Segment);  //search by bit-fields
25      If  not InMalwareFile  then { SegmentColl.delete(Segment); continue; }
26      }
27
28  SegmentsInMalwareOnly = 0;
29  ForEach Segment in SegmentColl do
30      { SegmentsInMalwareOnly  = SegmentsInMalwareOnly +1; }
31
32  Malware_Classfication_Result = SegmentsInMalwareOnly > ThreatSegmentThreshold;
```

(Ex. 1005, Figure 2).

272. Tahan also describes the feature selection/reduction process during training, which a POSITA would have understood includes selectively turning on or off features of a feature vector, as a common technique. Specifically, Tahan describes that a filtering approach to selecting or reducing features during training is useful for efficiency and generalizing capability (e.g., handling larger feature sets). (Ex. 1005, p. 3). Likewise, during classification of a file as benign

or malware, Tahan follows a similar process of segmenting the file, calculating meta-data about the segments and filtering the segments, and classifying based on the filtered segments. (Ex. 1005, pp. 8-11, Figure 1). Thus Tahan discloses feature selection during training, and makes it clear that these processes were a common practice throughout the process and would have been known to a POSITA.

273. For example, Tahan explains regarding training, "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. 1005, p. 2 ("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 limitation 1[e].) As discussed above, a POSITA would have understood that "a vector containing the features extracted" refers to a feature vector with 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 features (static data points)). (*See id.*; *see also* Ex. 1005, p. 3). 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., feature selection (feature reduction). This process discards a feature of a feature vector such that 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)). In the actual use of this trained classifier illustrated in Figure 1 of Tahan and described thereafter, the Mal-ID classifier also analyzes the meta-data of features and filters the features accordingly before classifying those features. (Ex. 1005, pp. 9-11).

274. In another example, Tahan discloses using feature sorting to turn on and off features during training 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. 1005, p. 3)), 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).

275. 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. 1005, p. 3, emphasis added).

276. Likewise, Tahan discloses using the "Gain Ratio feature selection" process later during classification of files using the Mal-ID algorithm. Specifically, Tahan states: "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. 1005, p. 13, emphasis added).

This Gain Ratio technique is a filtering technique as described in Menahem and in the background of Tahan. (Ex. 1006, p. 5 and Ex. 1005, p. 3 (both citing Mitchell, 1997 for gain ratio in selection of features)).

277. Menahem also discloses feature selection during training. 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. 1006, p. 5).

278. Menahem uses a similar approach to Tahan and explicitly defines 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. 1006, p. 5).

279. 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. 1006, p. 5).

280. 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".

281. 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. 1006, p. 6, emphasis added). This same gain ratio method is applied later in Menahem during the evaluation process (classification) to reduce the features evaluated in the data set to a computationally manageable amount.

282. Specifically, Menahem states:

> Unfortunately, due to processing power and memory limitations we could not run our ensemble experiments on the entire datasets and we had to reduce the datasets in order to complete the experiments in reasonable time. Therefore, for the evaluation process we had produced a new dataset from the original dataset by choosing randomly 33% of the instances (i.e., files)

> in the original datasets and we applied the Gain Ratio feature selection method (Mitchell, 1997) on the n-gram-based and PE datasets. As a result, each ensemble scheme in our experiment was tested upon 10 datasets.
> (Ex. 1006, p. 7).

283. To the extent that any of these approaches disclosed by Tahan-Menahem for turning features on or off are not explicitly disclosed to be during the classification process, a POSITA would have found it obvious to apply them during the classification process based on Forman. Forman taught a feature scaling process[5] used to turn features on or off (e.g., by "eliminating" a feature) during classification. According to Forman, the process is similar ("substantially similar" or "mirrors") the same process used during training:

> One potential type of processing performed by pre-processing section 14 is feature selection [during training], i.e., selecting only those features that are most predictive for use in classifying new items.
> (Ex. 1018, 3:11-14).

> Once classifier 18 has been generated, it can be ***used to classify unlabeled items 22***. Each such item 22 is input into a ***pre-processing section 24 that preferably performs processing that is identical or at least substantially similar to that of pre-processing section 14***…
> (Ex. 1018, 3:56-60, emphasis added).

> Next, in step 93 a score is calculated for each of the training samples. In the preferred embodiments, the score comprises the BNS score referenced above. However, as also noted above, in other embodiments a different score is calculated. In certain sub-embodiments, all features are removed that have received a feature score that is zero (or very close to Zero based on a pre-defined threshold), because feature values multiplied by Zero are always zero and therefore cannot help discriminate positives from negatives.
> (Ex. 1018, 6:66-7:8).

> FIG. 6 is a flow diagram illustrating a process for classifying an item according to a representative embodiment of the present invention. ***As will become apparent below, several of the steps of the preferred process for classifying an item mirror those described above for training the classifier 18.***

[5] This feature scaling process of Forman is not present in the '642 publication by the same author (US 2010/0082642 to Forman et al.), which was cited by the Examiner during prosecution.

> Initially, in step 102 an item 22 to be classified is obtained. As with the training samples 12, the item 22 preferably is represented by a set of values corresponding to a feature set 38, although in this case there generally is of course no pre-identified classification label. ***In certain embodiments, the feature set 38 has been modified from an original feature set, e.g., by eliminating certain features.***
> (Ex. 1018, 7:56-67, emphasis added).

> In step 105, the feature values for the item 22 are scaled based on the scores that were obtained in step 93. This step, as well as the considerations pertaining to it, preferably are similar to step 95, discussed above. More preferably, the identical scaling that was performed in the training step 95 (used for the training of the classifier that will be used in step 107) is performed in this step 105.
> (Ex. 1018, 8:7-13).

284. A POSITA would have understood based on these disclosures that the weighting and scaling used to remove or eliminate features during training would be supplied and used for similar elimination of features during classification as disclosed. It would have been obvious to a POSITA to combine Forman with Tahan and use Forman's feature scaling during classification with Tahan's classification methods. Thus, at least in combination with Forman, a POSITA would have found it obvious to apply any of Tahan's training-based feature reduction/selection to the classification process.

285. Additionally, Forman's disclosures describe turning features on or off based on the value of the static data point being within a range. It explains that features can first be bounded to a certain range. Forman discloses: "if an item to be classified 22 has a feature value that fall outside the previously known range [Vmin, Vmax], then its value is clipped to fall within this range before proceeding with the scaling." (Ex. 1018, 8:13-17). A POSITA would have understood that because Forman also teaches eliminating features based on scaling that such features could be eliminated if they dropped below Vmin or rose above Vmax depending on the configuration. In other words, values lower than Vmin or higher than Vmax would be turned off

(depending on the configuration). Similarly, values between Vmin and Vmax could be turned on (depending on the configuration).

286. It also would have been obvious to a POSITA to combine Forman with Menahem and Tahan to use Forman's feature scaling during classification with Menahem's classification methods. Thus, at least in combination with Forman, a POSITA would have found it obvious to apply any of the training-based feature reduction/selection in Tahan-Menahem to the classification process.

287. Moreover, a POSITA would have found it obvious to directly use Forman's teachings on feature scaling/elimination during classification time in combination with Tahan and Menahem. As already explained above, Tahan already turns features on or off during classification with the entropy measure of segments. As explained in Section IX.A., a POSITA would have recognized this process to be similar to Forman's scaling/elimination. As Forman explains, initial feature sets are often over inclusive:

> It often will be the case that ***the number of features included in the original feature set is over-inclusive***, e.g., including some features that are not very predictive of the desired classification category. There are several reasons that this situation can occur. First, one often wants to increase the likelihood that the feature set is able to adequately characterize the input items, and ***at the outset it sometimes is not possible to know which features will prove to be most predictive***, so it often is better to err on the side of over-inclusion. In addition, ***in order to minimize processing requirements, the same feature set sometimes will be generated and used for multiple different purposes, so the resulting feature set has some features that are not particularly appropriate for the specific task at hand***.
> (Ex. 1018 2:59-3:5).

288. Forman teaches that in order to deal with the over inclusivity of an initial feature set (e.g., during training), classification can scale down or even eliminate features, which directly addresses the problem of computational limits described in Menahem above.

> FIG. 6 is a flow diagram illustrating a process for classifying an item according to a representative embodiment of the present invention. As will become apparent below, several of the steps of the preferred process for classifying an item mirror those described above for training the classifier 18.
>
> Initially, in step 102 an item 22 to be classified is obtained. As with the training samples 12, the item 22 preferably is represented by a set of values corresponding to a feature set 38, although in this case there generally is of course no pre-identified classification label. ***In certain embodiments, the feature set 38 has been modified from an original feature set, e.g., by eliminating certain features***.
> (Ex. 1018, 7:56-67, emphasis added).

289. While Tahan identifies a specific feature type (data with high entropy) that can be disabled based on range during classification time, A POSITA would have understood that Forman's general scaling and elimination is more broadly applicable to almost any set of data.

290. As also noted in Section IX.A., in Menahem, the choice was made to reduce the size of the dataset: "due to processing power and memory limitations we could not run our ensemble experiments on the entire datasets and we had to reduce the datasets in order to complete the experiments in reasonable time." A POSITA would have been motivated to apply Forman's teachings of dynamically reducing features in order to test on a larger dataset.

291. Thus, a POSITA using Forman in combination with Tahan and/or Menahem, would have found it obvious to selectively turn on or off any of the features of the feature vector based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range. I have already explained in Section IX.A. why a POSITA would have been motivated to combine Tahan, Menahem, and Forman, and I incorporate this analysis by reference.

292. Thus, Tahan in view of Menahem and Forman discloses this element. Thus, Tahan in view of Menahem in further view of Forman discloses limitation 1[f].

7. Limitation 1[g]

293. Limitation 1[g] of the ’869 patent reads:

> *providing the generated feature vector to one or more support vector machines to build a probabilistic model that indicates whether the executable file fits into one of the categories of files.*

294. Tahan in view of Menahem in further view of Forman discloses limitation 1[g] of the ’869 patent. Tahan discloses that the use of support vector machines to generate a classifier that identifies harmful executable files (malware) (“indicates whether the executable file fits into one of the categories of files”) 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. 1005, pp. 2-3, emphasis added).

295. As discussed above (e.g., in Section V.B.), a POSITA would have known that the classification component produced by different types of SVM techniques can either be binary (i.e., classify the input as one thing or another) or provide a probabilistic output that attempts to quantify the likelihood of the category of the input. (*See* Section V.B., regarding “SVM can produce probabilistic outputs instead of strict categorical classifications.”). A POSITA would have understood that when the output of this latter type of SVM is probabilistic, the classifier is a probabilistic model for indicating “whether the executable file fits into one of the categories of files.”

296. As I explained in my overview of the reference, Menahem uses a mix of different machine learning techniques 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, which may be weighted based on performance at each level of processing. (Ex. 1006, pp. 4-5). 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. 1006, p. 2). 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. 1006, p. 2: "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.).

297. In the ensemble classifier disclosed by Menahem, when support vector machines are used as a component, the support vector machines are also being used to generate a probabilistic classifier component which, as explained above, is a probabilistic model. The composite classifier is created by aggregating results of multiple individual base classifiers (which are usually built with different techniques) in order to produce a more accurate classification than any individual classifier. (*See* Ex. 1006, p. 4: "A test instance is first classified by each of the base-classifiers. These classifications are fed into a meta-level training set from which a meta-classifier is produced. This classifier (Meta-classifier) combines the different predictions into a final one."). The composite classifier, which includes the SVM-generated classifier, is also a probabilistic model that identifies whether the executable file is malicious or benign. As explained in the preceding paragraph, one approach is to use the "Majority Voting" algorithm to combine the results of multiple classifiers (such as SVM classifiers). The number of votes for a particular outcome is a probabilistic model. Thus, a POSITA would have understood that Tahan in view of Menahem discloses multiple ways in which "support vector machines" are used "to build a probabilistic model that indicates whether the executable file fits into one of the categories of files."

298. I have already explained in Section IX.A. why a POSITA would have been motivated to combine Tahan and Menahem (e.g., including why a POSITA would have been 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. 1006, p. 2).

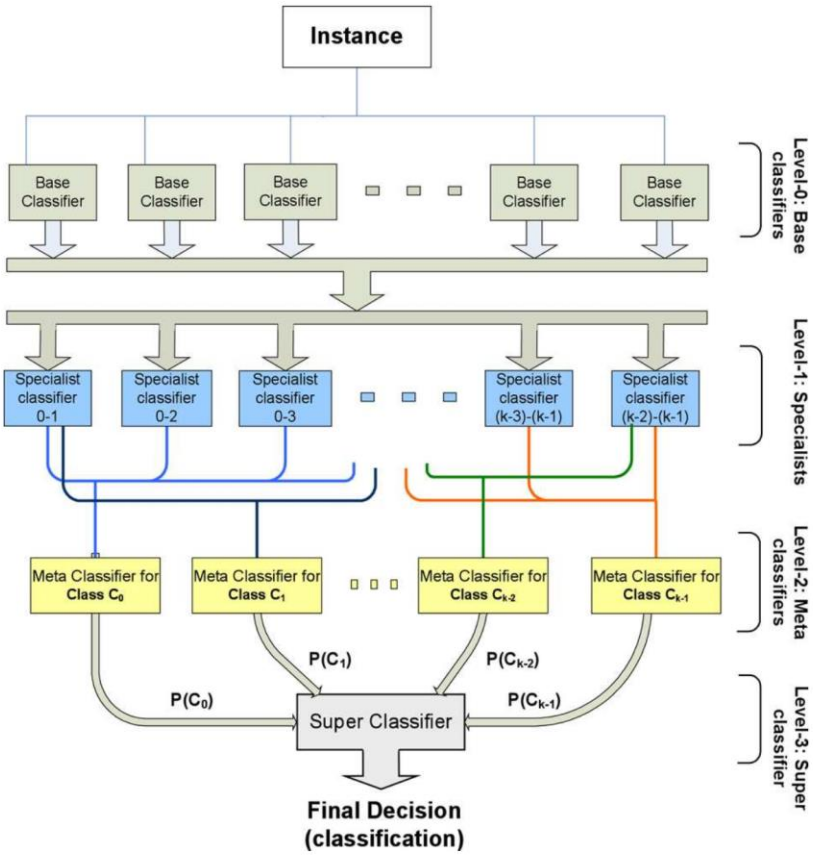
299. Tahan already teaches (and provides motivation for) combining Mal-ID with other classifiers and finds that the results were better in certain circumstances. Tahan explains that "[w]e attempted to improve the Mal-ID basic method by using Mal-ID features with various classifiers…" (Ex. 1005, p. 11). Specifically, Tahan combines Mal-ID with three other classification technologies known as "Rotation Forest," "C4.5", and a "multilayer perception." (Ex. 1005, p. 13). Tahan further identifies that other classifiers were known in the art to be used for classifying malware, including Support Vector Machines (SVM). (*See* Ex. 1005, pp. 3-5: "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.).

300. 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. 1005, p. 28). 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. 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.*).

301. To the extent that neither Tahan nor Menahem explicitly disclose using support vector machines (which is not conceded), it would have been obvious to a POSITA in view of Forman. (*See id.*). Forman explicitly uses Support Vector Machines. As Forman explains, "In the preferred embodiments of the invention, training section 16 implements Support Vector Machine (SVM) training in order to output its classifier 18." (Ex. 1018, 3:46-49). As I have already explained, both Tahan and Menahem discuss the wide use of support vector machines (SVM) in the research art and would have been motivated to incorporate Forman's teachings. *Id.*

302. 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. 1006, p. 4).

303. 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).

304. 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. 1006, p. 12). 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. 1005, p. 4). 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. 1006, p. 5).

305. Tahan also noted that SVM outperformed the C4.5 model. "In the evaluation the SVM model performed slightly better then the C4.5 model." (Ex. 1005, p. 4). Notably, both Tahan and Menahem used the C4.5 model. (*See, e.g.*, Ex. 1005, p. 12 and Ex. 1006, p. 5). 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).

306. As discussed above, the composite ensemble classifier with SVM, as disclosed in Tahan-Menahem in view of Forman, is a classifier to evaluate executable files. For example, Tahan discloses detecting malware in a new application and a file of the application (i.e., an executable file). (Ex. 1005, pp. 8, 10). That is, the classifiers produced by Tahan and/or Menahem in further view of Forman operate on executable files. As Tahan explains, the ML techniques are used to create a "classifier that is able to distinguish malware from benign files." (Ex. 1005, p. 2). Neither Tahan nor Menahem expresses any limitation of the files that can be evaluated by the classifier, and a POSITA would have known that the classifier can provide an output on any file from which a compatible feature vector can be generated. (*See* Section V.A.).

Trend Micro, Inc.
EX1003-120

307. Moreover, to the extent that a probabilistic model is not explicit in Tahan or Menahem, Forman teaches a probabilistic model and teaches providing the feature vector of the executable file to the SVM classifier. See Figure 1 of Forman annotated below:

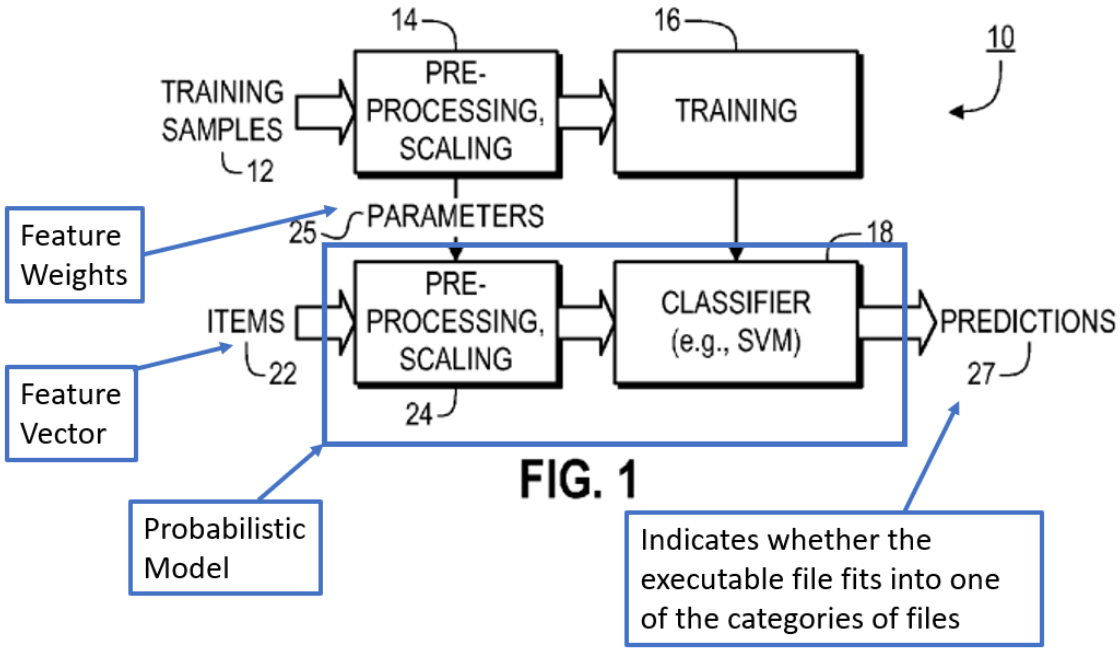


FIG. 1

308. Specifically, and as explained in Section VIII.D., Forman combines a weight model with a feature vector at classification time by an SVM classifier.

> Once classifier 18 has been generated, it can be used to classify unlabeled items 22. Each such item 22 is input into a pre-processing section 24 that preferably performs processing that is identical or at least substantially similar to that of pre-processing section 14 [(i.e., during training)]… Classifier 18 then operates on the modified feature set output from pre-processing section 24 to provide output predictions 27 of the underlying ground truth labels for the subject items 22… As indicated above, one aspect of the present invention is feature scaling prior to use of a feature set for training and/or classification purposes.
> (Ex. 1018, 3:56-4:11).

309. The feature selection is based on weighting (scoring):

> However, in the event feature selection is used in conjunction with the techniques of the present invention, and based solely on the limited

> experimentation performed to date, the currently preferred embodiments employ a different scoring technique for feature selection (e.g., Information Gain, as discussed in more detail below)…
> (Ex. 1018, 3:24-30).

310. Forman explains that a probabilistic model can be used to weight the features: "in the preferred embodiments module 40 scores each feature using a score defined… [using] F^{-1} is the inverse standard Normal cumulative probability distribution function." (Ex. 1018, 4:43-62). (*See also* Section VIII.D.). A POSITA would have understood that by applying a probabilistic weighting technique to the inputs of the SVM classifier Forman is teaching a probabilistic model for classifying the executable.

311. Thus, Tahan in view of Menahem in further view of Forman builds a probabilistic model together with identifying an executable file to be evaluated. Thus, Tahan in view of Menahem in further view of Forman discloses limitation 1[g].

312. Therefore, I conclude that Tahan in view of Menahem in further view of Forman renders Claim 1 obvious to a POSITA.

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

313. Claim 2 of the '869 patent, which depends from Claim 1, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 2 reads,

> *The computer-implemented method according to claim 1, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.*

314. As I explained above with respect to limitation 1[e], these features ("static data points") are extracted from executable files. Moreover, Tahan teaches that its techniques (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. 1005, p. 1, emphasis added; *see also* p. 27). 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.*)

315. Moreover, Tahan explains that its techniques and processes work with executable files in general or "in the Wild." (*See id.*; *see also id.*, p. 10 ("The input for the Mal-ID method is an unclassified executable file of any size."), p. 27 ("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 '869 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.
> (Ex. 1001, 2:40-53, emphasis added).

316. 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.*)

317. Therefore, I conclude that Tahan in view of Menahem in further view of Forman renders Claim 2 obvious to a POSITA.

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

318. Claim 3 of the '869 patent, which depends from Claim 1, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 3 reads,

> *The computer-implemented method according to claim 1, wherein the one or more support vector machines builds the probabilistic model by performing data analysis and pattern recognition on the one or more feature vectors.*

319. Tahan describes its process of building an SVM classifier using data mining approaches:

> Instead of relying on unpacking methods that may fail, Dai et al. (2009) proposed a malware-detection system, based on a virtual machine, to reveal and capture the needed features. The system constructs classification

> models ***using common data mining approaches***. First, both malware and benign programs are executed inside the virtual machine and the instruction sequences are collected during runtime. ***Second, the instruction sequence patterns are abstracted. Each sequence is treated as a feature.*** Next, a feature selection process in [sic] performed. In the last stage a classification model is built. In the evaluation the SVM model performed slightly better then [sic] the C4.5 model.
> (Ex. 1005, p. 4, emphasis added).

320. It should be noted that when Tahan refers to "common data mining approaches," it refers to approaches well known in the art at the time. A reference cited by both Tahan and Menahem, a 2005 textbook by Witten and Frank titled "Data Mining: Practical Machine Learning Tools and Techniques", describes "data mining" in terms that a POSITA would have understood teach the same concepts as the claimed "data analysis and pattern recognition":

> In data mining, the data is stored electronically and the search is automated—or at least augmented—by computer. Even this is not particularly new. Economists, statisticians, forecasters, and communication engineers have long worked with the idea that ***patterns in data can be sought automatically, identified, validated, and used for prediction***. ***What is new is the staggering increase in opportunities for finding patterns in data.*** The unbridled growth of databases in recent years, databases for such everyday activities as customer choices, brings data mining to the forefront of new business technologies. It has been estimated that the amount of data stored in the world's databases doubles every 20 months, and although it would surely be difficult to justify this figure in any quantitative sense, we can all relate to the pace of growth qualitatively. As the flood of data swells and machines that can undertake the searching become commonplace, the opportunities for data mining increase. As the world grows in complexity, overwhelming us with the data it generates, data mining becomes our only hope for elucidating the patterns that underlie it. ***Intelligently analyzed data is a valuable resource.*** It can lead to new insights, and, in commercial settings, to competitive advantages.
>
> Data mining is about solving problems by analyzing data already present in databases. Suppose, to take a well-worn example, the problem is fickle customer loyalty in a highly competitive marketplace. A database of customer choices, along with customer profiles, holds the key to this problem. Patterns of behavior of former customers can be analyzed to identify distinguishing characteristics of those likely to switch products and those likely to remain loyal. Once such characteristics are found, they can be put to work to identify present customers who are likely to jump ship.

> This group can be targeted for special treatment, treatment too costly to apply to the customer base as a whole. More positively, the same techniques can be used to identify customers who might be attracted to another service the enterprise provides, one they are not presently enjoying, to target them for special offers that promote this service. In today's highly competitive, customer-centered, service-oriented economy, data is the raw material that fuels business growth—if only it can be mined.
>
> ***Data mining is defined as the process of discovering patterns in data.*** The process must be automatic or (more usually) semiautomatic. The patterns discovered must be meaningful in that they lead to some advantage, usually an economic one. The data is invariably present in substantial quantities.
> (Ex. 1014, pp. 4-5, emphasis added).

321. Moreover, to the extent that a probabilistic model is not explicit in Tahan or Menahem, Forman teaches a providing a probabilistic model and the feature vector of the executable file to the SVM classifier. Specifically, and as explained in Section VIII.D., Forman combines a weight model with a feature vector at classification time by an SVM classifier.

> Once classifier 18 has been generated, it can be used to classify unlabeled items 22. Each such item 22 is input into a pre-processing section 24 that preferably performs processing that is identical or at least substantially similar to that of pre-processing section 14 [(i.e., during training)]… Classifier 18 then operates on the modified feature set output from pre-processing section 24 to provide output predictions 27 of the underlying ground truth labels for the subject items 22… As indicated above, one aspect of the present invention is feature scaling prior to use of a feature set for training and/or classification purposes.
> (Ex. 1018, 3:56-4:11).

322. The feature selection is based on weighting (scoring):

> However, in the event feature selection is used in conjunction with the techniques of the present invention, and based solely on the limited experimentation performed to date, the currently preferred embodiments employ a different scoring technique for feature selection (e.g., Information Gain, as discussed in more detail below)…
> (Ex. 1018, 3:24-30).

323. Forman explains that a probabilistic model can be used to weight the features: "in the preferred embodiments module 40 scores each feature using a score defined… [using] F^{-1} is

the inverse standard Normal cumulative probability distribution function." (Ex. 1018, 4:43-62). (*See also* Section VIII.D.).

324. A POSITA would have understood that by applying a probabilistic weighting technique to the inputs of the SVM classifier Forman is teaching a probabilistic model for classifying the executable.

325. A POSITA would have recognized that Tahan's disclosures, in referring to "common data mining approaches" without elaborating on specific approaches, would rely on techniques known in the art at the time, such as those described in Witten, and that such techniques would include the generic terms "data analysis and pattern recognition". In combination with Forman's probabilistic models for the weighted input, the data analysis and pattern recognition construct the probabilistic model.

326. Thus, a POSITA would have understood Tahan's disclosures of building SVM models to teach the claimed process of "build[ing] the probabilistic model by performing data analysis and pattern recognition on the one or more feature vectors."

327. Thus, Tahan in view of Menahem in further view of Forman renders Claim 3 obvious to a POSITA.

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

328. Claim 4 of the '869 patent, which depends from Claim 1, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 4 reads,

> *The computer-implemented method according to claim 1, wherein the probabilistic model indicates whether the executable file is harmful.*

329. I have already explained with respect to limitation 1[g] that Tahan in view of Menahem in further view of Forman teaches "build[ing] a probabilistic model that indicates

whether the executable file fits into one of the categories of files", such "categories of files including harmful files and benign files". (*See* my analyses of limitations 1[c] and 1[g]). Furthermore, Tahan and Menahem each disclose Claim 4.

330. Tahan discloses the use of machine learning models for classifying files as malicious or benign: "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." (Ex. 1005, p. 2).

331. Menahem also discloses this concept. Its method uses ensemble classifiers whose predictions are based on the results of the component ("base") classifiers. "A test instance is first classified by each of the base-classifiers. These classifications are fed into a meta-level training set from which a meta-classifier is produced. This classifier (Meta-classifier) combines the different predictions into a final one. … Consequently the meta-classifier predictions reflect the true performance of base-level learning algorithms. Stacking performance can be improved by using output probabilities for every class label from the base-level classifiers. It has been shown that with stacking, the ensemble performs (at best) comparably to selecting the best classifier from the ensemble by cross-validation (Dzeroski and Zenko, 2004)." (Ex. 1006, p. 4).

332. I have already discussed in Section IX.A. that a POSITA would have been motivated to combine Tahan and Menahem. In particular, because both references teach using machine learning models to classify files as malicious or benign, a POSITA would have expected a high likelihood of success in such a combination, especially with reference to this claim.

333. Thus, I conclude that Tahan and Menahem each disclose Claim 4.

F. Claim 8 is rendered obvious by Tahan in view of Menahem in further view of Forman

334. Claim 8 of the ’869 patent, which depends from Claim 1, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 8 reads,

> *The computer-implemented method according to claim 1, wherein the plurality of static data points represent predefined character strings in the executable file.*

335. As I explained with reference to Claim 1 (especially limitations 1[e]), Tahan alone or combined with Menahem discloses extracting from executable files various “static data points” described by the ’869 patent.

336. 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. 1005, p. 4, 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.

337. It should be noted that Tahan’s “segments” are intended to represent meaningful portions of the executable file. As I discussed above with reference to limitations 1[b] and 1[c], Tahan teaches the use of a Common Function Library (CFL), which “contains data structures constructed from benign files”, and a Threat Function Library (TFL), which “contains data

structures constructed from malware without segments identified as benign". (Ex. 1005, p. 7). 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. 1005, p. 10). 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. 1005, p. 4, emphasis added). Thus, Tahan discloses extracting "static data points" that "represent predefined character strings in the executable file".

338. In another example, as I discussed with reference to the limitation 1[e] ("extracting, by the knowledge module, a plurality of static data points from the identified executable file"), Tahan discloses that it was common in the art to use PE header fields such as debug flags, CPU type, etc. (Ex. 1005, p. 2). 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.

339. 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.

340. 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. 1006, pp. 5-6).

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

342. 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. 1006, p. 6). 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. 1006, p. 6).

343. Although Forman does not explicitly discuss identifying binary malware, it does disclose processing of text strings such as the text strings analyzed by Tahan and Menahem. As Forman explains:

> For example, for purposes of classifying an input e-mail message to determine whether the message is spam or non-spam, the feature set in one embodiment of the invention pertains to a list of words, with each the data field for each feature (i.e., word in this example) intended to hold a binary value indicating whether the word is present in the e-mail message. In alternate embodiments, the feature set accommodates entry of an integer number for each word, indicating the number of occurrences of the word in the e-mail message. Various techniques for selecting a feature set, with the particular technique typically depending upon the particular classification problem, are discussed in the conventional literature.
> (Ex. 1018, 2:46-58).

344. I have already explained in Section IX.A. and above for this limitation why a POSITA would have been motivated to combine Tahan, Menahem, and Forman, 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.

345. 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

Trend Micro, Inc.
EX1003-132

decision. We weigh the individual opinions, and combine them to reach a final decision." (Ex. 1006, p. 2). Substituting an ensemble classifier for a single base classifier was known in the art, as discussed in prior art and in the '869 patent itself. (*See, e.g.*, Ex. 1001, 9:27-54; *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.

346. Furthermore, a POSITA would have applied Forman's teaching of feature scaling and classification-time elimination in order to tailor over-inclusive feature sets to individual problems and problem types.

347. Thus, Tahan in view of Menahem in further view of Forman render Claim 8 obvious to a POSITA.

G. Claim 10 is rendered obvious by Tahan in view of Menahem in further view of Forman

348. Claim 10 of the '869 patent is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

349. The following table compares the language of Claim 10 with that of Claim 1, with differences indicated in strikethrough and bold:

1[a]: A ~~computer-implemented method~~ comprising:	**10[a]:** A **system** comprising: **at least one memory; and** **at least one processor operatively connected with the memory and configured to perform operation of:**

1[b]: identifying, ~~by a knowledge module,~~ static data points that may be indicative of either a harmful or benign executable file;	**10[b]:** identifying static data points that may be indicative of either a harmful or benign executable file;
1[c]: associating, ~~by the knowledge module,~~ the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;	**10[c]:** associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;
1[d]: identifying an executable file to be evaluated;	**10[d]:** identifying an executable file to be evaluated;
1[e]: extracting, ~~by the knowledge module,~~ a plurality of static data points from the identified executable file;	**10[e]:** extracting a plurality of static data points from the identified executable file; **and**
1[f]: generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the	**10[f]:** generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the

plurality of static data points is within a predetermined range; and	plurality of static data points is within a predetermined range; and
1[g]: providing the generated feature vector to one or more support vector machines to build a probabilistic model that indicates whether the executable file fits into one of the categories of files.	**10[g]:** providing the generated feature vector to one or more support vector machines to build a probabilistic model that indicates whether the executable file fits into one of the categories of files.

350. Thus, the differences are limitation 10[a], the addition of the conjunction "and" at the end of limitation 10[e], and the omission of "by a/the knowledge module" in limitations 10[b], 10[c], and 10[e].

1. Limitation 10[a]

351. 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 '869 patent is considered limiting, Tahan in view of Menahem in further view of Forman discloses the preamble of Claim 10.

352. 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. 1005, p. 14). A POSITA would have understood that running a program 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).

353. Menahem discloses that its algorithms were "implemented in WEKA (Witten and Frank, 2005) in the JAVA programming language." (Ex. 1006, p. 5). 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).

354. 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 have been motivated to combine Tahan and Menahem and I incorporate this analysis by reference.

2. Limitations 10[b]—10[g]

355. Furthermore, the omission of "by a/the knowledge module" in limitations 10[b], 10[c], and 10[e] does not change my analysis compared with the similar limitations of Claim 1, so I incorporate herein by reference my analysis of limitations 1[b] through 1[g] for limitations 10[b] through 10[g].

356. Thus, I conclude that Tahan in view of Menahem in further view of Forman renders Claim 10 obvious to a POSITA.

H. Claim 11 is rendered obvious by Tahan in view of Menahem in further view of Forman

357. Claim 11 of the '869 patent, which depends from Claim 10, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 11 reads,

> *The system according to claim 10, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.*

358. The language of Claim 11 following "wherein" matches that of Claim 2. I therefore incorporate herein by reference my analysis on Claim 2 and conclude that Claim 11 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

I. Claim 12 is rendered obvious by Tahan in view of Menahem in further view of Forman

359. Claim 12 of the '869 patent, which depends from Claim 10, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 12 reads,

> *The system according to claim 10, wherein the one or more support vector machines builds the probabilistic model by performing data analysis and pattern recognition on the one or more feature vectors.*

360. The language of Claim 12 following "wherein" matches that of Claim 3. I therefore incorporate herein by reference my analysis on Claim 3 and conclude that Claim 12 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

J. Claim 13 is rendered obvious by Tahan in view of Menahem in further view of Forman

361. Claim 13 of the '869 patent, which depends from Claim 10, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 13 reads,

> *The system according to claim 10, wherein the probabilistic model indicates whether the executable file is harmful.*

362. The language of Claim 13 following "wherein" matches that of Claim 4. I therefore incorporate herein by reference my analysis on Claim 4 and conclude that Claim 13 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

K. Claim 14 is rendered obvious by Tahan in view of Menahem in further view of Forman

363. Claim 14 of the '869 patent, which depends from Claim 10, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 14 reads,

> *The system according to claim 10, wherein the plurality of static data points represent predefined character strings in the executable file.*

364. The language of Claim 14 following "wherein" matches that of Claim 8. I therefore incorporate herein by reference my analysis on Claim 8 and conclude that Claim 14 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

L. Claim 15 is rendered obvious by Tahan in view of Menahem in further view of Forman

365. Claim 15 of the '869 patent is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

366. The following table compares the language of Claim 15 with that of Claim 1, with differences indicated in strikethrough and bold:

1[a]: A ~~computer-implemented method~~ comprising:	**15[a]:** A **computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process** comprising:
1[b]: identifying, ~~by a knowledge module,~~ static data points that may be indicative of either a harmful or benign executable file;	**15[b]:** identifying static data points that may be indicative of either a harmful or benign executable file;

1[c]: associating, ~~by the knowledge module,~~ the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;	**15[c]:** associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;
1[d]: identifying an executable file to be evaluated;	**15[d]:** identifying an executable file to be evaluated;
1[e]: extracting, ~~by the knowledge module,~~ a plurality of static data points from the identified executable file;	**15[e]:** extracting a plurality of static data points from the identified executable file;
1[f]: generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and	**15[f]:** generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and

1[g]: providing the generated feature vector to one or more support vector machines to build a probabilistic model that indicates whether the executable file fits into one of the categories of files.	**15[g]:** providing the generated feature vector to one or more support vector machines to build a probabilistic model that indicates whether the executable file fits into one of the categories of files.

367. Thus, the differences are limitation 15[a] and the omission of "by a/the knowledge module" in limitations 15[b], 15[c], and 15[e].

1. Limitation 15[a]

368. I understand that the preamble of a patent claim is not always considered limiting. However, to the extent that the preamble of Claim 15 of the '869 patent is considered limiting, Tahan in view of Menahem in further view of Forman discloses the preamble of Claim 15.

369. I discussed with reference to limitation 10[a] that Tahan and Menahem both disclose algorithms stored in memory or storage devices that, when executed on a connected processor, teach the subsequent limitations. I incorporate that analysis herein by reference.

2. Limitations 15[b]—15[g]

370. Furthermore, the omission of "by a/the knowledge module" in limitations 15[b], 15[c], and 15[e] does not narrow the scope of the claim, so I incorporate herein by reference my analysis of limitations 1[b] through 1[g] for limitations 15[b] through 15[g].

371. Thus, I conclude that Tahan in view of Menahem in further view of Forman renders Claim 15 obvious to a POSITA.

M. Claim 16 is rendered obvious by Tahan in view of Menahem in further view of Forman

372. Claim 16 of the ’869 patent, which depends from Claim 15, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 16 reads,

> *The computer-readable storage device according to claim 15, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.*

373. The language of Claim 16 following “wherein” matches that of Claim 2. I therefore incorporate herein by reference my analysis on Claim 2 and conclude that Claim 16 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

N. Claim 17 is rendered obvious by Tahan in view of Menahem in further view of Forman

374. Claim 17 of the ’869 patent, which depends from Claim 15, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 17 reads,

> *The computer-readable storage device according to claim 15, wherein the plurality of static data points represent predefined character strings in the executable file.*

375. The language of Claim 17 following “wherein” matches that of Claim 8. I therefore incorporate herein by reference my analysis on Claim 8 and conclude that Claim 17 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

O. Claim 18 is rendered obvious by Tahan in view of Menahem in further view of Forman

376. Claim 18 of the ’869 patent is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

377. The following table compares the language of Claim 18 with that of Claim 1, with differences indicated in strikethrough and bold:

1[a]: A computer-implemented method comprising:	**18[a]:** A computer-implemented method comprising:
1[b]: identifying, ~~by a knowledge module,~~ static data points that may be indicative of either a harmful or benign executable file;	**18[b]:** identifying static data points that may be indicative of either a harmful or benign executable file;
1[c]: associating, ~~by the knowledge module,~~ the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;	**18[c]:** associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;
1[d]: identifying an executable file to be evaluated;	**18[d]:** identifying an executable file to be evaluated;
1[e]: extracting, ~~by the knowledge module,~~ a plurality of static data points from the ~~identified~~ executable file;	**18[e]:** extracting a plurality of static data points from the executable file;
1[f]: generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of	**18[f]:** generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of

categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and	categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the plurality of static data points is within a predetermined range; and
1[g]: ~~providing~~ the generated feature vector ~~to one or more support vector machines to build a probabilistic model that indicates~~ whether the executable file fits into one of the categories of files.	**18[g]: evaluating** the feature vector **using a machine learning model to determine** whether the executable file fits into one of the categories of files.

378. Thus, the differences are the omission of "by a/the knowledge module" in limitations 18[b], 18[c], and 18[e], the removal of "identified" from 18[e], and the most of 18[g].

1. Limitations 18[a]—18[f]

379. I have already discussed with reference to Claim 1 that Tahan in view of Menahem in further view of Forman renders obvious these claim limitations. Furthermore, the omission of "by a/the knowledge module" in limitations 18[b], 18[c], and 18[e] does not narrow the scope of the claim. In addition, the omission of "identified" in limitation 18[e] does not change the antecedent of "the executable file". Therefore, I incorporate herein by reference my analysis of limitations 1[b] through 1[g] for limitations 18[b] through 18[g].

2. Limitation 18[g]

380. Limitation 18[g] of the '869 patent reads:

evaluating the feature vector using a machine learning model to determine whether the executable file fits into one of the categories of files.

381. I have already discussed with reference to limitation 1[g] that Tahan, Menahem, and Forman disclose the use of a particular machine learning technique, support vector machines, for evaluating the feature vector derived from an executable file. For example, Tahan discloses that the use of support vector machines to generate a classifier that identifies harmful executable files (malware) ("whether the executable file fits into one of the categories of files") 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. 1005, pp. 2-3, emphasis added). Support vector machines are machine learning algorithms. (*See* Section V.B.).

382. Moreover, Forman discloses Support Vector Machines (SVM) as an example of machine learning. As Forman explains, "In the preferred embodiments of the invention, training section 16 implements Support Vector Machine (SVM) training in order to output its classifier 18." (Ex. 1018, 3:46-49). Additionally, Forman notes that binary classifiers, like SVM, are often extended to multi-classification (classifying into more than two labels) using multiple classifiers. A single classification of the many (1-of-n) can be determined "by decomposing the 1-of-n problem, pitting each class against the others." (Ex. 1018, 2:23-24). Alternatively, it is possible to

identify many matching classes out of the set of total classes (n-of-m) "by applying m independent binary classifiers to each item." (Ex. 1018, 2:26).

383. In addition, all three references are explicit about the interchangeability of additional machine learning techniques, which produce "machine learning models", including those disclosed by the '869 patent in Claims 20-25.

384. For example, Tahan discloses that many machine learning techniques are capable of producing malware classifications, including "ensemble methods":

> 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: artificial neural networks (ANNs) (Bishop, 1995), decision tree (DT) learners (Quinlan, 1993), nave-Bayes [sic] (NB) classifiers (John and Langley, 1995), Bayesian networks (BN) (Pearl, 1987), support vector machines (SVMs) (Joachims, 1999), k-nearest neighbor (KNN) (Aha et al., 1991), voting feature intervals (VFI) (Demiröz and Güvenir, 1997), OneR classifier (Holte, 1993), Adaboost (Freund and Schapire, 1999), random forest (Breiman, 2001), and other ensemble methods (Menahem et al., 2009; Rokach, 2010).
> (Ex. 1005, p. 3).

385. Menahem discloses creating ensemble classifiers using a variety of different machine learning techniques (which Menahem refers to as "inducers"), including many of those listed by Tahan:

> In the experiment we had used five different inducers as our base-classifiers: C4.5 (Quinlan, 1993), KNN (Kibler, 1991), VFI (Guvenir, 1997), OneR (Holte, 1993) and Naïve Bayes (John and Langley, 1995). Each inducer belongs to different family of classifiers. For example, C4.5 is related to Decision Trees, KNN belongs to Lazy classifiers, OneR to Rules, Naïve-Bayes to Bayes classifiers and VFI to general.
> (Ex. 1006, p. 5).

386. Furthermore, Forman teaches the use of the Bayesian algorithm, but clarifies that other types of algorithms can be used suitably within its teachings:

> However, in alternate embodiments a nonlinear SVM, a Naïve Bayes ***or any other type of classifier is trained***, preferably using a supervised or semi-supervised training technique..
> (Ex. 1018, 7:42-44, emphasis added).

387. In summary, Tahan, Menahem, and Forman each teach the suitability of multiple algorithms for this problem domain.

388. In addition, Tahan and Menahem teach the use of ensemble classifiers, and Menahem provides motivation for using such ensemble classifiers, particularly when composed of base classifiers that employ unique algorithms:

> The outcome of this observation is that one should use multiple different classifiers that will classify a new instance predicting its real class (like a group of experts) and then combining all the results, using an intelligent combination method, into one final result. Hopefully, this "super-classifier" will outperform each of the individual classifiers and classify new instances with higher accuracy by learning the strength and weakness of each classifier.
> (Ex. 1006, p. 2).

389. Thus, I conclude that Tahan in view of Menahem in further view of Forman renders Claim 18 obvious to a POSITA.

P. Claim 19 is rendered obvious by Tahan in view of Menahem in further view of Forman

390. Claim 19 of the '869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 19 reads,

> *The computer-implemented method according to claim 18, wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.*

391. The language of Claim 19 matches that of Claim 2, aside from the reference to the claim from which it depends. I therefore incorporate herein by reference my analysis on Claim 2

and conclude that Claim 19 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

Q. Claim 20 is rendered obvious by Tahan in view of Menahem in further view of Forman

392. Claim 20 of the ’869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 20 reads,

> *The computer-implemented method according to claim 18, wherein the machine learning model comprises an artificial neural network.*

393. Tahan explicitly teaches using an artificial neural network: “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: artificial neural networks (ANNs)…” (Ex. 1005, p. 3).

394. A POSITA would have understood from Tahan’s disclosures that many compatible classification techniques exist that are suitable for classifying malware, including artificial neural networks. A POSITA would have been motivated to consider using an artificial neural network when implementing the claimed method, and, based on the research available at the time, would have expected a high likelihood of success when using such a technique.

395. Therefore, Tahan in view of Menahem in further view of Forman renders Claim 20 obvious to a POSITA.

R. Claim 21 is rendered obvious by Tahan in view of Menahem in further view of Forman

396. Claim 21 of the ’869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 21 reads,

> *The computer-implemented method according to claim 18, wherein the machine learning model comprises a support vector machine.*

Trend Micro, Inc.
EX1003-147

397. I have already discussed with reference to limitation 1[g] that Tahan in view of Menahem renders obvious the use of a support vector machine as the machine learning model for the disclosed malware analysis. I incorporate my analysis of that limitation herein by reference.

398. Thus, I conclude that Tahan in view of Menahem in further view of Forman renders Claim 21 obvious to a POSITA.

S. Claim 22 is rendered obvious by Tahan in view of Menahem in further view of Forman

399. Claim 22 of the '869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 22 reads,

> *The computer-implemented method according to claim 18, wherein the machine learning model comprises a machine learning decision tree.*

400. Tahan explicitly teaches using decision tree techniques: "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: … decision tree (DT) learners…" (Ex. 1005, p. 3).

401. Menahem also explicitly teaches using a decision tree technique, C4.5: "In the experiment we had used five different inducers as our base-classifiers: C4.5 … Each inducer belongs to different family of classifiers. For example, C4.5 is related to Decision Trees …" (Ex. 1006, p. 5).

402. A POSITA would have understood from both Tahan's and Menahem's disclosures that many compatible classification techniques exist that are suitable for classifying malware, including decision trees. A POSITA would have been motivated to consider using decision trees when implementing the claimed method, and, based on the research available at the time, would have expected a high likelihood of success when using such a technique.

403. Therefore, Tahan in view of Menahem in further view of Forman renders Claim 22 obvious to a POSITA.

T. Claim 23 is rendered obvious by Tahan in view of Menahem in further view of Forman

404. Claim 23 of the '869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 23 reads,

> *The computer-implemented method according to claim 18, wherein the machine learning model comprises a Bayesian network.*

405. Tahan explicitly teaches using Bayesian techniques: "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: … nave-Bayes [sic] (NB) classifiers (John and Langley, 1995), Bayesian networks (BN) (Pearl, 1987) …" (Ex. 1005, p. 3).

406. Menahem also explicitly teaches using Bayesian techniques: "In the experiment we had used five different inducers as our base-classifiers: … Naïve Bayes (John and Langley, 1995). Each inducer belongs to different family of classifiers. For example, C4.5 is related to Decision Trees, KNN belongs to Lazy classifiers, OneR to Rules, Naïve-Bayes to Bayes classifiers and VFI to general." (Ex. 1006, p. 5).

407. Forman also discloses the use of Bayesian machine learning approaches for its models:

> However, in alternate embodiments a nonlinear SVM, a ***Naïve Bayes*** or any other type of classifier is trained, preferably using a supervised or semi-supervised training technique.
> (Ex. 1018, 7:42-44, emphasis added).

408. A POSITA would have understood from these disclosures of Tahan, Menahem, and Forman that many compatible classification techniques exist that are suitable for classifying

malware, including Bayesian techniques such as Naïve-Bayes, Multi-Naïve Bayes, and Bayesian networks. Menahem in particular is explicit about using multiple "families" of classifiers in its ensemble. A POSITA would have been motivated to consider using Bayesian techniques when implementing the claimed method, and, based on the research available at the time, would have expected a high likelihood of success when using such techniques.

409. Therefore, Tahan in view of Menahem in further view of Forman renders Claim 22 obvious to a POSITA.

U. Claim 24 is rendered obvious by Tahan in view of Menahem in further view of Forman

410. Claim 24 of the '869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 24 reads,

> *The computer-implemented method according to claim 18, wherein evaluating the feature vector using the machine learning model comprises evaluating the feature vector using support vector processing.*

411. The term "support vector processing" as used in the '869 patent specification refers to the use of support vector machines techniques (see Ex. 1001, 13:7-10: "In examples, operation 306 further comprises evaluating the feature vector using linear or nonlinear support vector processing to determine a classification for the executable file"). Thus, a POSITA would have understood the term "support vector processing" to teach the same techniques as "support vector machines".

412. As I have already discussed with reference to limitations 1[g] and 18[g], Tahan and Menahem and Forman each disclose the use of support vector machines. A POSITA would have understood from such disclosures that support vector machines represent one of many compatible classification techniques suitable for malware classification. A POSITA would have been

motivated to consider using support vector machines when implementing the claimed method, and, based on the research available at the time, would have expected a high likelihood of success when using such a technique.

413. Thus, I conclude that Tahan in view of Menahem in further view of Forman renders Claim 24 obvious to a POSITA.

V. Claim 25 is rendered obvious by Tahan in view of Menahem in further view of Forman

414. Claim 25 of the '869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 25 reads,

> *The computer-implemented method according to claim 18, wherein the machine learning model comprises a clustering model.*

415. Tahan explicitly teaches using k-nearest neighbor (KNN) techniques: "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: … k-nearest neighbor (KNN)…" (Ex. 1005, p. 3).

416. Menahem also explicitly teaches using KNN techniques: "In the experiment we had used five different inducers as our base-classifiers: … KNN (Kibler, 1991)…" (Ex. 1006, p. 5).

417. A POSITA would have known that KNN is a technique used in clustering, a machine learning technique for discovering related groups of data within a data set. For example, as I explained in Section V.C., some clustering models operate in multiple steps first to quantify the relatedness of data points (e.g., by constructing sparse sub-graphs) and then to discover clusters based on those sub-graphs. A clustering model such as CHAMELEON uses KNN to generate sub-graphs that inform its cluster selection process: "CHAMELEON finds the initial sub-clusters using a graph partitioning algorithm to partition the k-nearest neighbor graph of the data set into a large

number of partitions such that the edge-cut, i.e., the sum of the weight of the edges that straddle partitions, is minimized." (Ex. 1012, p. 9; *see also* Section V.C.).

418. A POSITA would have understood from both Tahan's and Menahem's disclosures that many compatible classification techniques exist that are suitable for classifying malware, including clustering techniques. A POSITA would have been motivated to consider using clustering techniques when implementing the claimed method, and, based on the research available at the time, would have expected a high likelihood of success when using such techniques.

419. Thus, I conclude that Tahan in view of Menahem in further view of Forman renders Claim 25 obvious to a POSITA.

W. Claim 29 is rendered obvious by Tahan in view of Menahem in further view of Forman

420. Claim 29 of the '869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman. Claim 29 reads,

> *The computer-implemented method according to claim 18, wherein the plurality of static data points represent predefined character strings in the executable file.*

421. The language of Claim 29 matches that of Claim 8, aside from the reference to the claim from which it depends. I therefore incorporate herein by reference my analysis on Claim 8 and conclude that Claim 29 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

X. Claims 32 and 46 are rendered obvious by Tahan in view of Menahem in further view of Forman

422. Claims 32 and 46 of the '869 patent are rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

423. The following table compares the language of Claims 32 and 46 with that of Claim 18, with differences indicated in strikethrough and bold:

18[a]: A ~~computer-implemented method~~ comprising:	**32[a]:** A **computer-readable storage device containing instructions, that when executed on a processor, cause the processor to execute a process** comprising:	**46[a]:** A **system** comprising: **at least one memory; at least one processor configured to perform operation of:**
18[b]: identifying static data points that may be indicative of either a harmful or benign executable file;	**32[b]:** identifying static data points that may be indicative of either a harmful or benign executable file;	**46[b]:** identifying static data points that may be indicative of either a harmful or benign executable file; **and**
18[c]: associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;	**32[c]:** associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files;	**46[c]:** associating the identified static data points with one of a plurality of categories of files, the plurality of categories of files including harmful files and benign files; **and**

18[d]: identifying an executable file to be evaluated;	**32[d]:** identifying an executable file to be evaluated;	**46[d]: at least one processor configured to perform operation of:** identifying an executable file to be evaluated;
18[e]: extracting a plurality of static data points from the executable file;	**32[e]:** extracting a plurality of static data points from the executable file;	**46[e]:** extracting a plurality of static data points from the executable file;
18[f]: generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the	**32[f]:** generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the	**46[f]:** generating a feature vector from the plurality of static data points using a classifier trained to classify the static data points based on training data, the training data comprising files known to fit into one of the plurality of categories of files, wherein one or more features of the feature vector are selectively turned on or off based at least in part on evaluation of whether a value of one of the

plurality of static data points is within a predetermined range; and	plurality of static data points is within a predetermined range; and	plurality of static data points is within a predetermined range; and
18[g]: evaluating the feature vector using a machine learning model to determine whether the executable file fits into one of the categories of files.	**32[g]:** evaluating the feature vector using a machine learning model to determine whether the executable file fits into one of the categories of files.	**46[g]:** evaluating the feature vector using a machine learning model to determine whether the executable file fits into one of the categories of files.

424. Thus, the differences are the changes in 32[a].

1. Limitations 32[a] and 46[a]

425. I discussed with reference to limitation 10[a] that Tahan and Menahem both disclose algorithms stored in memory or storage devices that, when executed on a connected processor, teach the subsequent limitations. I incorporate that analysis herein by reference.

2. Limitations 32[b]—32[g] and limitations 46[b]—46[g]

426. The addition of the word "and" at the end of 46[b] and 46[c] does not change the meaning of either limitation. In addition, the repeat of "at least one processor configured to perform operation of" at the beginning of 46[d] does not change the meaning of that limitation, since the preceding two limitations had an identical phrase. The remaining limitations of Claims 32 and 46 are identical to the corresponding limitations of Claim 18. Therefore, I incorporate my analysis of limitations 18[b] through 18[g] herein by reference.

Trend Micro, Inc.
EX1003-155

427. Thus, I conclude that Tahan in view of Menahem in further view of Forman renders Claims 32 and 46 obvious to a POSITA.

Y. Claims 33-39, 43, 47-53, and 57 are rendered obvious by Tahan in view of Menahem in further view of Forman

428. The remaining claims of the '869 patent correspond to dependent claims of Claim 18 and are rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

429. The following table compares the relevant language of Claims 33-39, 43, 47-53, and 57 with that of Claims 19-25, and 29, with differences indicated in bold:

19: … wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.	**33:** … wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.	**47:** … wherein the plurality of static data points are extracted without decrypting or unpacking the executable file.
20: … wherein the machine learning model comprises an artificial neural network.	**34:** … wherein the machine learning model comprises an artificial neural network.	**48:** … wherein the machine learning model comprises an artificial neural network.
21: … wherein the machine learning model comprises a support vector machine.	**35:** … wherein the machine learning model comprises a support vector machine.	**49:** … wherein the machine learning model comprises a support vector machine.
22: … wherein the machine learning model comprises a	**36:** … wherein the machine learning model comprises a	**50:** … wherein the machine learning model comprises a

machine learning decision tree.	machine learning decision tree.	machine learning decision tree.
23: … wherein the machine learning model comprises a Bayesian network.	**37:** … wherein the machine learning model comprises a Bayesian network.	**51:** … wherein the machine learning model comprises a Bayesian network.
24: … wherein evaluating the feature vector using the machine learning model comprises evaluating the feature vector using support vector processing.	**38:** … wherein evaluating the feature vector using the machine learning model comprises evaluating the feature vector using support vector processing.	**52:** … wherein evaluating the feature vector using the machine learning model comprises evaluating the feature vector using support vector processing.
25: … wherein the machine learning model comprises a clustering model.	**39:** … wherein the machine learning model comprises a clustering model.	**53:** … wherein the machine learning model comprises a clustering model.
29: … wherein the plurality of static data points represent predefined character strings in the executable file.	**43:** … wherein the plurality of static data points represent predefined character strings in the executable file.	**57:** … wherein the plurality of static data points represent predefined character strings in the executable file.

430. As noted in the table above, the relevant language of these claims are the same across the three groups. Thus, I incorporate my analyses of Claims 19-25, and 29 herein by

Trend Micro, Inc.
EX1003-157

reference and conclude that Claims 33-39, 43, 47-53, and 57 are rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman.

X. GROUND 2: CLAIMS 5-7, 9, 26-28, 30-31, 40-42, 44-45, 54-56, AND 58-59 ARE RENDERED OBVIOUS BY TAHAN IN VIEW OF MENAHEM IN FURTHER VIEW OF FORMAN IN FURTHER VIEW OF SCHULTZ

431. As I describe in greater detail hereinbelow, a POSITA would have found Claims 5-7, 9, 26-28, 30-31, 40-42, 44-45, 54-56, and 58-59 rendered obvious by Tahan in view of Menahem in further view of Schultz.

A. Motivation to Combine Tahan, Menahem, Forman, and Schultz

432. A POSITA would have been motivated to combine the teachings of Tahan, Menahem, Forman, and Schultz. I have already provided a motivation to combine Tahan, Menahem and Forman in Section IX.A. I incorporate the analysis of that section by reference.

433. A POSITA would have been further motivated to combine the teachings of Tahan, Menahem, and Forman 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."

434. Schultz 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. 1007, [0045]-[0055], [0068]; *see also* my analysis for Claim 1). Forman, also a production system, likewise uses ML classifiers for classifying email. (Ex. 1018, 2:39-58). As described in Section IX.A., the

combination of Forman with Tahan and Menahem provides improved efficiency with feature scaling and eliminating features at classification time. Thus, a POSITA would have understood that Tahan, Menahem, and Schultz are in the same field and that their methods/systems are compatible. Like Forman, 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 Forman, Tahan and Menahem could be substituted for, modified by, and/or enhanced by the classifiers described in Schultz.

435. For example, Schultz provides very practical deployment instructions, such as how to identify malicious email attachments at their ingress to the network (*see* my analysis of Claim 6 below) and how to handle re-training the model in order to improve prediction accuracy, using borderline cases subsequently confirmed by a human operator as malicious or benign (*see* my analysis of Claim 9 below). 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, Menahem, and Forman in practice.

B. Claim 5 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz

436. Claim 5 of the ’869 patent, which depends from Claim 1, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz. Claim 5 reads,

> *The computer-implemented method according to claim 1, wherein the executable file is identified in response to a detected condition.*

437. Schultz discloses a component of its system which filters and subsequently classifies email attachments (e.g., “executable attachments”), in response to their arrival on the system and before they can be executed:

> The process 100 of detecting malicious emails in accordance with the invention is illustrated in FIG.8. ***A first step is to receive the emails at the server (step 102).*** In the exemplary embodiment, the mail server is Sendmail. Procmail is a publicly available program that processes e-mail messages received by the server and looks for particular information in the headers or body of the message, and takes action on what it finds.
>
> ***Subsequently, the emails are filtered to extract attachments or other components from the email (step 104).*** The executable attachments may then be saved to a file. In the exemplary embodiment, html_trap.procmail, a commonly available routine, has been modified to include a call to a novel routine, parser3, which performs the functions of filtering attachments. The routine parser3 includes a call to the routine extractAttachments, for example, which extracts executable attachments and other items of the email and saves them to a file…
> (Ex. 1007, [0096]-[0097], emphasis added).
>
> ***Features in the executable attachment are extracted (step 106), and those features are subsequently analyzed and used to classify the executable attachment as malicious or benign.***
> (Ex. 1007, [0098], emphasis added).

438. The ’869 patent gives some examples of identifying an executable file “in response to a detected condition”:

> In addition to collecting and managing information to train a learning classifier, the knowledge component **102** is used to evaluate an executable file and extract/collect static data points for evaluation of the file by the learning classifier before the file is executed. In one example, the system automatically identifies a new file (e.g., unknown file) for evaluation. As examples, identification of an executable file for evaluation may occur in examples such as: ***when a download of a file is requested, while a download is being performed, evaluation of streaming data, when a new file is detected as attempting to execute (e.g., potentially unwanted file), and before an unknown file or file containing executable code that was not previously checked attempts to execute***, among other examples.
> (Ex. 1001, 7:1-14, emphasis added).

439. A POSITA would have understood Schultz’s disclosures regarding intercepting email attachments as they are received at the server to teach, at a minimum, a concept similar to that of the ’869 patent’s teaching of identifying an executable file “while a download is being

performed", because the server is receiving data (i.e., "downloading" data) from an external source (e.g., the mail server sending the email).

440. I have already discussed in Section X.A. that a POSITA would have been motivated to combine Tahan with Menahem, Forman, and Schultz. In particular, Schultz teaches a method for identifying items to be evaluated after the training process is complete, a necessary step for implementing a classification system in production. Because Schultz teaches classification methods compatible with those described in Tahan, Menahem, and Forman, a POSITA would have been motivated to combine Schultz's identification techniques with the teaching of Tahan, Menahem, and Forman, and would have expected a high likelihood of success in such combination.

441. Thus, Tahan in view of Menahem in further view of Forman in further view of Schultz renders Claim 5 obvious to a POSITA.

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

442. Claim 6 of the '869 patent, which depends from Claim 5, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz. Claim 6 reads,

> *The computer-implemented method according to claim 5, wherein the detected condition is user request for a file download.*

443. Tahan discloses the need for monitoring files downloaded from the Internet, as they may be malicious: "Nowadays the use of the Internet has become an integral part of modern life and Internet browsers are downloading to users a wide variety of content, including new computer software. One consequence of this widespread use is that many computer systems are vulnerable to and infected with malware—malicious software." (Ex. 1005, p. 1).

444. Schultz teaches monitoring email attachments received at a mail server (which a POSITA would have understood to constitute a "file download"). I described Schultz's teachings about such file downloads with reference to Claim 5, and I incorporate that analysis herein by reference.

445. Schultz teaches that its system may operate on various machines in a network, including the email server or a client computer: "The system may resides [sic] on the server of a computer or on a host or client of the computer system to receive emails before they are forwarded to users of the system." (Ex. 1007, [0104]).

446. A POSITA would have understood that a user requesting the download of their email messages from a mail server to their client computer would encompass the claimed "user request for a file download". A POSITA would further recognize that Schultz's processing occurs in response to such a detected condition.

447. I have already discussed in Section X.A. that a POSITA would have been motivated to combine Tahan with Menahem, Forman, and Schultz. In particular, Schultz teaches a method for identifying items to be evaluated after the training process is complete, a necessary step for implementing a classification system in production. Because Schultz teaches classification methods compatible with those described in Tahan, Menahem, and Forman, a POSITA would have been motivated to combine Schultz's identification techniques with the teaching of Tahan, Menahem, and Forman and would have expected a high likelihood of success in such combination.

448. Thus, Tahan in view of Menahem in further view of Forman in further view of Schultz renders Claim 6 obvious to a POSITA.

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

449. Claim 7 of the ’869 patent, which depends from Claim 5, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz. Claim 7 reads,

> *The computer-implemented method according to claim 5, wherein the detected condition is the detection of a new file attempting to execute.*

450. Schultz aims to identify malicious executables: “An object of the present invention is to provide a technique for predicting a classification of an executable file as malicious or benign which is not dependent upon the format of the executable.” (Ex. 1007, [0017]). In particular, Schultz conducts its analyses without executing the files: “All of the information about the binary is obtained from the program header without executing the unknown program but by examining the static properties of the binary…” (Ex. 1007, [0044]).

451. Schultz recognizes that email attachments may be automatically executed by email clients and underscores the need for systems that prevent such execution when the attachments are malicious. For example:

> Procmail is a mail processing utility which runs under UNIX, and which filters email; and sorts incoming email according to sender, subject line, length of message, keywords in the message, etc. Procmail's pre-existent filter provides the capability of detecting active-content HTML tags to protect users who read their mail from a web browser or HTML-enabled mail client. ***Also, if the attachment is labeled as malicious, the system “mangles” the attachment name to prevent the mail client from automatically executing the attachment.***
> (Ex. 1007, [0014]).

452. Indeed, as discussed in Section V.F., a POSITA would have understood that some email attachments are capable of self-execution when opened in a “conventional email client”:

> In a broad sense, the several embodiments disclosed herein provide systems and methods that are more secure from attack by ***viruses, worms, or Trojan***

> ***programs. Since these undesired programs may be embedded as self-executing code in non-ASCII-based email systems (e.g., HTML-based email systems),*** the generation of a modified email message without the embedded functions, which are amenable to the embedding of self-executing codes, may greatly reduce the opportunity for such attacks.
> (Ex. 1017, 2:42-50, emphasis added).
>
> For those instances, if the original email message is corrupted by a harmful self-executing code, such as a virus, worm, or Trojan program, then ***the conventional email client would typically launch that self-executing code.***
> (Ex. 1017, 5:35-39, emphasis added).

453. A POSITA would have understood an auto-executing attachment, as discussed by Schultz and as is known in the art, is attempting to execute immediately upon arrival in the email client. Thus, a POSITA would also understand any scanning and blocking of such a file by Schultz constitutes detecting a new file (i.e., the new attachment) attempting to execute.

454. For example, Schultz teaches quarantining attachments or emails containing attachments via procmail routines (e.g., html-trap.procmail) as a method for preventing such automatic execution:

> 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. 1007, [0103], emphasis added).
>
> The results of this analysis may be sent to the email interface 232, which reintegrates filtered email back into normal email traffic 300, and which may send the model generator 240 (described below) each attachment to be analyzed further. If the program was considered malicious by the rule evaluator 230, the email interface 232 may add warnings to the email or ***quarantine the email***. In the exemplary embodiment, the routine html-trap.procmail performs this function.
> (Ex. 1007, [0108], emphasis added).

455. Furthermore, Schultz is clear that the goal of its approach is "detecting ***unknown*** binaries," a teaching similar to the claimed "***new file*** attempting to execute." (*See* Ex. 1007, [0123], describing that the goal of its algorithm is "detecting unknown binaries" and that its

approach for including data about misclassified binaries can be done "without lowering the accuracy of the algorithm in detecting unknown binaries.").

456. In addition to these disclosures by Schultz, a POSITA would have been familiar with the state of the art of anti-malware techniques at the time, including those developed as least as early as 1999 that included monitoring execution attempts of new files in order to detect and prevent malware. (*See, e.g.,* Section V.D. regarding "on-access" scanning in Dr. Solomon's Anti-Virus Toolkit). A POSITA would have known that such monitoring was an important characteristic of functioning malware detection and prevention systems, and would have been motivated to include such techniques in systems they developed. To the extent the disclosures of Schultz do not teach the claimed "detection of a new file attempting to execute", a POSITA would have been motivated to combine Schultz's teachings with these well-known techniques of monitoring execution attempts of new files, and they would have found it obvious to do so. Furthermore, because Schultz discloses and builds upon existing techniques (e.g., procmail) that seek to prevent automatic execution of malicious executable files, a POSITA would have expected a high likelihood of success in combining such techniques with the teachings of Schultz.

457. Thus, I conclude that Tahan in view of Menahem in further view of Forman in further view of Schultz would have rendered Claim 7 obvious to a POSITA.

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

458. Claim 9 of the '869 patent, which depends from Claim 1, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz. Claim 9 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.

459. Schultz teaches that it is desirable to identify borderline cases, have an expert properly classify them, and use the results to improve the detection model:

> A further step may be to identify the programs as borderline (step 110). Borderline executables are defined herein as programs that have similar probabilities of being benign and malicious (e.g., 50% chance it is malicious, and 50% chance it is benign: 60% chance malicious and 40% chance benign: 45% chance malicious and 55% chance benign, etc.). Due to the similar probabilities, borderline executables are likely to be mislabeled as either malicious or benign. Since borderline cases could potentially lower the detection and accuracy rates by being misclassified, ***it is desirable to identify these borderline cases, properly classify them as malicious or benign, and update the classification rule set to provide increased accuracy to the detection of malicious executables.*** The larger the data set that is used to generate models, then the more accurate the detection models will be. To execute this process, the system identifies programs as borderline using the criteria described below, and archives the borderline cases. At periodic intervals, the system sends the collection of these borderline cases to a central server, by the system administrator. Once at a central repository, such as data repository 244, these binaries can then be analyzed by experts to determine whether they are malicious or not, and subsequently included in the future versions of the detection models.
> (Ex. 1007, [0100], emphasis added).

460. Schultz then teaches how to incorporate these properly-classified borderline cases into the model:

> 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. 1007, [0110]).

461. 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.

462. 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. 1005, p. 17). 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 IX.A.).

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

F. Claim 26 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz

464. Claim 26 of the '869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz. Claim 26 reads,

> *The computer-implemented method according to claim 18, wherein the executable file is identified in response to a detected condition.*

465. The language of Claim 26 matches that of Claim 5, aside from the reference to the claim from which it depends. I therefore incorporate herein by reference my analysis on Claim 5 and conclude that Claim 26 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz.

G. Claim 27 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz

466. Claim 27 of the ’869 patent, which depends from Claim 26, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Schultz. Claim 27 reads,

> *The computer-implemented method according to claim 26, wherein the detected condition is user request for a file download.*

467. The language of Claim 27 matches that of Claim 6, aside from the reference to the claim from which it depends. I therefore incorporate herein by reference my analysis on Claim 6 and conclude that Claim 27 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz.

H. Claim 28 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz

468. Claim 28 of the ’869 patent, which depends from Claim 26, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz. Claim 28 reads,

> *The computer-implemented method according to claim 26, wherein the detected condition is the detection of a new file attempting to execute.*

469. The language of Claim 28 matches that of Claim 7, aside from the reference to the claim from which it depends. I therefore incorporate herein by reference my analysis on Claim 7 and conclude that Claim 28 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz.

I. Claim 30 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz

470. Claim 30 of the ’869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz. Claim 30 reads,

> *The computer-implemented method according to claim 18, wherein a determination of whether the executable file is harmful is used to retrain the classifier.*

471. The language of Claim 30 matches that of Claim 9, aside from the reference to the claim from which it depends. I therefore incorporate herein by reference my analysis on Claim 9 and conclude that Claim 30 is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz.

J. Claim 31 is rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz

472. Claim 31 of the ’869 patent, which depends from Claim 18, is rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz. Claim 31 reads,

> *The computer-implemented method according to claim 18, further comprising preventing execution of the executable file based on classifying the executable file as harmful.*

473. 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. First:

> Subsequently, the emails are filtered to extract attachments or other components from the email (step 104). The executable attachments may then be saved to a file. In the exemplary embodiment, html_trap.procmail, a commonly available routine, has been modified to include a call to a novel routine, parser3, which performs the functions of filtering attachments. The routine parser3 includes a call to the routine extractAttachments, for example, which extracts executable attachments and other items of the email and saves them to a file…
> (Ex. 1007, [0097]).

474. Thereafter, "[t]he 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. 1007, [0103]; *see also id.,* [0108]). A POSITA would have understood that quarantining an executable file results in preventing it from executing. For example, Schultz teaches that Procmail has a mechanism for preventing execution of malicious attachments: "Procmail's pre-existent filter provides the capability of detecting active-content HTML tags to protect users who read their mail from a web browser or HTML-enabled mail client. Also, if the attachment is labeled as malicious, the system 'mangles' the attachment name to prevent the mail client from automatically executing the attachment." (Ex. 1007, [0014]).

475. As discussed in Section V.F., a POSITA would have understood that some email attachments are capable of self-execution and must be properly filtered by a system, such as that taught by Schultz, to prevent an attack by a malicious attachment:

> In a broad sense, the several embodiments disclosed herein provide systems and methods that are more secure from attack by ***viruses, worms, or Trojan programs. Since these undesired programs may be embedded as self-executing code in non-ASCII-based email systems (e.g., HTML-based email systems),*** the generation of a modified email message without the embedded functions, which are amenable to the embedding of self-executing codes, may greatly reduce the opportunity for such attacks.
> (Ex. 1017, 2:42-50, emphasis added).

476. Indeed, "conventional email clients" may run such malware automatically:

> For those instances, if the original email message is corrupted by a harmful self-executing code, such as a virus, worm, or Trojan program, then ***the conventional email client would typically launch that self-executing code.*** (Ex. 1017, 5:35-39, emphasis added).

477. Therefore, a POSITA would have understood that the disclosures of Schultz would tend to prevent such automatic execution when malware is detected, through mechanisms such as quarantining the attachment or "mangling" the attachment.

478. Thus, I conclude that Tahan in view of Menahem in further view of Forman in further view of Schultz renders Claim 31 obvious to a POSITA.

K. Claims 40-42, 44-45, 54-56, and 58-59 are rendered obvious by Tahan in view of Menahem in further view of Forman in further view of Schultz

479. The remaining claims of the '869 patent correspond to dependent claims of Claim 18 and are rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz.

480. The following table compares the relevant language of Claims 40-42, 44-45, 54-56, and 58-59 with that of Claim 26-28 and 30-31, with differences indicated in bold:

26: … wherein the executable file is identified in response to a detected condition.	**40:** … wherein the executable file is identified in response to a detected condition.	**54:** … wherein the executable file is identified in response to a detected condition.
27: … wherein the detected condition is user request for a file download.	**41:** … wherein the detected condition is user request for a file download.	**55:** … wherein the detected condition is user request for a file download.
28: … wherein the detected condition is the detection of a	**42:** … wherein the detected condition is the detection of a	**56:** … wherein the detected condition is the detection of a

new file attempting to execute.	new file attempting to execute.	new file attempting to execute.
30: … wherein a determination of whether the executable file is harmful is used to retrain the classifier.	**44:** … wherein a determination of whether the executable file is harmful is used to retrain the classifier.	**58:** … wherein a determination of whether the executable file is harmful is used to retrain the classifier.
31: … further comprising preventing execution of the executable file based on classifying the executable file as harmful.	**45:** … **wherein the process further comprises** preventing execution of the executable file based on classifying the executable file as harmful.	**59:** … further comprising **at least one processor configured to perform operation of:** preventing execution of the executable file based on classifying the executable file as harmful.

481. As noted in the table above, all but the last of these claims are the same across the three groups. The difference between Claims 45 and Claim 31 is not relevant, and the difference between Claim 59 and Claim 31 does not change its meaning (since the dependent claim already requires "at least one processor configured to perform operation of…").

482. The language of Claims 46 and 59 that requires "at least one processor" to perform particular operations may be read to indicate that the operations are performed on a system with at least a single processor. This reading anticipates multicore or multiprocessor systems capable of performing these operations but does not require a dedicated processor for each of the enumerated operations. To the extent that the language "at least one processor" is interpreted to mean that the

enumerated operations run on distinct processors, or that the enumerated operations be capable of running on multiple processors, a POSITA would have understood that common practice when writing software for multicore or multiprocessor systems is to allow two or more portions of the software to run at the same time (e.g., by writing multiple applications that run independently on different processors, or through the use of threads within a single application). (*See* Section V.E.). A POSITA implementing the teachings of Tahan, Menahem, Forman, and Schultz would have been motivated to employ such parallelization in order to take advantage of the additional speed and effectiveness afforded by such techniques. (*See id.).*

483. Thus, I incorporate my analyses of Claims 19-31 herein by reference and conclude that Claims 33-45 and 47-59 are rendered obvious to a POSITA by Tahan in view of Menahem in further view of Forman in further view of Schultz.

XI. THERE ARE NO OBJECTIVE INDICIA OF NONOBVIOUSNESS

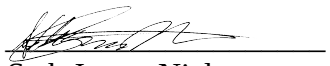
484. I am not aware of any objective indicia of non-obviousness related specifically to the ’869 patent. Indeed, I am not aware of any evidence of commercial success, long felt but unsolved need, failure of others, industry praise, or unexpected results specifically related to the ’869 patent.

XII. CONCLUSION

485. 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 September 4, 2023

By: 
Seth James Nielson