

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SOPHOS LTD. and SOPHOS INC.,
Petitioners

v.

WEBROOT INC.,
Patent Owner.

Case No. IPR2023-00528
U.S. Patent No. 10,599,844

Petition for *Inter Partes* Review of U.S. Patent No. 10,599,844

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

TABLE OF CONTENTS

TABLE OF CONTENTS.....	i
TABLE OF EXHIBITS	iv
TABLE OF ABBREVIATIONS	vi
I. INTRODUCTION	1
II. MANDATORY NOTICES	3
A. Real Parties-in-Interest 37 C.F.R. § 42.8(b)(1)	3
B. Related Matters Under 37 C.F.R. § 42.8(b)(2).....	3
1. Current Litigation.....	3
2. Administrative Proceedings	3
C. Lead and Back-Up Counsel.....	3
D. Service of Information Under 37 C.F.R. § 42.8(b)(4)	5
III. PAYMENT OF FEES	5
IV. REQUIREMENTS OF <i>INTER PARTES</i> REVIEW	5
A. Standing.....	5
B. Identification of Challenge and Relief Requested	6
1. U.S. Patent Appl. Pub. No. 2015/0213376 (“Ideses”) (Ex. 1005).....	7
2. Japanese Patent Appl. Pub. No. 2012027710A (“Mori”) (Ex. 1006).....	7
C. How the Challenged Claims Are to be Construed Under 37 C.F.R. § 42.104(b)(3)	7
1. The terms relating to whether “character strings” are “within a predetermined range” (claims 1, 10, and 17).....	9
2. “within a predetermined range” (claims 1, 10, and 17).....	10
3. “support vector processing” (claims 1, 10, and 17).....	10
D. How the Challenged Claims Are Unpatentable Under 37 C.F.R. § 42.104(b)(4).....	10
E. Supporting Evidence Under 37 C.F.R. § 42.104(b)(5).....	11
V. THRESHOLD REQUIREMENT FOR <i>INTER PARTES</i> REVIEW	11

VI.	PERSON OF ORDINARY SKILL IN THE ART.....	11
VII.	STATE OF THE ART	12
	1. Malware	12
	2. Machine Learning / Support Vector Machines.....	12
	3. Software Encryption and Packing (Compression).....	13
	4. Static and Dynamic Malware Analysis	13
VIII.	OVERVIEW OF THE '844 PATENT.....	14
	A. Summary of the Prosecution History of the '844 Patent	15
IX.	DETAILED CHALLENGE.....	17
	A. Ground 1: Claims 1–20 Are Rendered Obvious By Ideses	17
	1. Prior Art Overview.....	17
	2. Analysis.....	18
	B. Ground 2: Claims 1, 8–10, and 17 Are Rendered Obvious By Mori.....	47
	1. Prior Art Overview.....	47
	2. Analysis.....	49
	C. Ground 3: Claims 1–20 Are Rendered Obvious Over Ideses in View of Mori	61
	1. Overview of Ground	61
	2. Analysis of Unpatentability and Motivation to Combine.....	61
X.	PTAB DISCRETION SHOULD NOT PRECLUDE INSTITUTION.....	64
	A. PTAB Should Not Exercise Its Discretion to Deny Institution Under <i>Fintiv</i>	64
	1. Factor 1: Likelihood of Stay.....	65
	2. Factor 2: District Court Schedule	65
	3. Factor 3: Petitioners' Investment in IPR Outweighs Forced Investment in Litigation to Date.....	68
	4. Factor 4: The Petition Raises Unique Issues	69
	5. Factor 5: Institution Will Provide the Board an Opportunity to Invalidate Claims That Later Could Be Reasserted Against Others	70

6.	Factor 6: Other Circumstances Support Institution	70
B.	PTAB Should Not Exercise its Discretion To Deny Institution Under <i>Becton</i> and <i>Advanced Bionics</i>	71
XI.	CONCLUSION	71
	APPENDIX: CHALLENGED CLAIM LISTING.....	75

TABLE OF EXHIBITS

Exhibit No.	Document
1001	U.S. Patent No. 10,599,844 (“the ’844 Patent”)
1002	File History for U.S. Patent No. 10,599,844
1003	Defendant Sophos Ltd.’s Stipulation Regarding Invalidity Contentions for U.S. Patent No. 10,599,844
1004	Declaration of Dr. Gene Tsudik
1005	U.S. Patent Appl. Pub. No. 2015/0213376 (“Ideses”)
1006	Japanese Patent Appl. Pub. No. 2012027710A (“Mori”) and Certified Translation Thereof
1007	Defendants’ Opening Claim Construction Brief, D.I. 86, <i>Webroot Inc. et al. v. AO Kaspersky Lab et al.</i> , No. 6:22-cv-00243 (W.D. Tex. Oct. 28, 2022)
1008	Order Granting Fourth Amended Scheduling Order, D.I. 160, <i>Webroot Inc. et al. v. AO Kaspersky Lab et al.</i> , No. 6:22-cv-00243 (W.D. Tex. Jan. 22, 2023)
1009	Wang <i>et al.</i> , <i>Detecting Unknown Malicious Executables Using Portable Executable Headers</i> , Fifth International Joint Conference on INC, IMS and IDC, 278-84 (2009)
1010	Michael Sikorski & Andrew Honig, <i>Practical Malware Analysis</i> (2012)
1011	Murugiah Souppaya & Karen Scarfone, <i>Guide to Malware Incident Prevention and Handling for Desktops and Laptops</i> , NIST Spec. Pub. 800-83 (Rev. 1) (July 2013)
1012	Monnappa K A, <i>Learning Malware Analysis</i> (2018)
1013	V. Kecman, <i>Support Vector Machines – An Introduction</i> , StudFuzz 177, 1–47 (2005)
1014	<i>Virus Bulletin</i> (Nov. 1990)
1015	U.S. Patent No. 8,709,924
1016	U.S. Patent No. 9,465,940
1017	D. Devi & S. Nandi, <i>Detection of Packed Malware</i> , in Proceedings of the First International Conference on Security of

Exhibit No.	Document
	Internet of Things (SecurIT '12), Association for Computing Machinery, New York, NY, USA, 22–26 (2012)
1018	Excerpt from Transcript of Motions Hearing, <i>Webroot Inc. et al. v. AO Kaspersky Lab et al.</i> , No. 6:22-cv-00243 (W.D. Tex. Dec. 9, 2022)

TABLE OF ABBREVIATIONS

Abbreviation	Term
'844 Patent	U.S. Patent No. 10,599,844
AIA	America Invents Act
Challenged Claims	Claims 1–20 of U.S. Patent No. 10,599,844
Ex.	Exhibit
Fig.	Figure
Ideses	U.S. Patent Appl. Pub. No. 2015/0213376 (Ex. 1005)
IPR	<i>inter partes</i> review
Mori	Japanese Patent Appl. Pub. No. 2012027710A (Ex. 1006)
Patent Owner	Webroot, Inc.
Petitioners	Sophos Ltd. and Sophos Inc.

I. INTRODUCTION

Pursuant to 35 U.S.C. § 311, Petitioners submit this Petition for *inter partes* review of U.S. Patent No. 10,599,844 (Ex. 1001), and seek cancellation of claims 1–20 as unpatentable under 35 U.S.C. § 103.

The '844 Patent relates to “threat detection of executable files”—specifically, through a method whereby a “plurality of static data points” are extracted from an executable file “without decrypting or unpacking the executable file.” Ex. 1001, Abstract. The extracted static data points are analyzed by “applying a classifier” that is trained from “known” malicious, benign, and unwanted executable files. *Id.* Based on the results of the analysis, a determination is made as to whether the executable file is harmful. *Id.*

Every one of the claimed features of this allegedly novel malware detection process was disclosed in or would have been obvious in light of U.S. Patent Appl. Pub. No. 2015/0213376 (“Ideses”) (Ex. 1005), which was not considered by the Examiner during prosecution of the '844 Patent and is a focus of this Petition. Ideses explicitly discloses the extraction of “static meta data”—*i.e.*, a “plurality of static data points” as required by the claims of the '844 Patent—and the training and application of a classifier to detect malware. *See* Ex. 1005, Abstract, [0089], [0134]–[0135]. Ideses discloses that its malware detection can be performed “off-line”—that is, “without decrypting or unpacking the executable file,” as required by the

independent claims of the '844 Patent. *Id.*, [0090]. Ideses likewise discloses two additional features required by each of the independent claims, specifically, the use of “support vector processing” (*i.e.*, the use of a “support vector machine”), as well as “selectively” turning features of a classifier on or off based on whether a value of “one or more static data points from the plurality of extracted static data points is within a predetermined range.” *See, e.g., id.*, [0074], [0096], [0104]–[0110].

The claimed features in the independent claims of the '844 Patent were also disclosed in (or would have been obvious in light of) Japanese Patent Appl. Pub. No. 2012027710A (“Mori”) (Ex. 1006), which was likewise not considered by the Examiner during prosecution of the '844 Patent. Like Ideses, Mori discloses a malware detection method that “focuses on readable strings contained in software files, collects information (characteristic vectors) that characterizes malware, such as messages contained in programs that constitute malware, host names, execution command names, function names and library names used for communication,” and “compares this content with strings contained in files being inspected to detect malware.” Ex. 1006, [0023]. Mori’s method is performed without using “decryption, disassembly, and debuggers,” as such techniques “are not suitable for use in detecting communications containing malware in real time.” *Id.*, [0007].

This Petition demonstrates that there is a reasonable likelihood that Petitioners will prevail on at least one of the claims challenged in the petition based upon prior

art references that the USPTO did not have before it during prosecution. Therefore, IPR should be instituted, and claims 1–20 should be canceled as unpatentable under 35 U.S.C. §§ 102 and 103 over the prior art references identified in this Petition.

II. MANDATORY NOTICES

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

Sophos Ltd., and Sophos, Inc. are the real parties-in-interest.

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

1. Current Litigation

The '844 Patent is presently the subject of litigation in five suits brought by Patent Owner against Sophos Ltd. and four other defendants in: *Webroot Inc, et al. v. Sophos Ltd.*, No. 6:22-cv-240 (W.D. Tex.); *Webroot, Inc. et al. v. Trend Micro, Inc.*, No. 6:22-cv-239; *Webroot, Inc. et al. v. CrowdStrike Holdings, Inc. et al.*, No. 6:22-cv-241; *Webroot, Inc. et al. v. AO Kaspersky Lab*, No. 6:22-cv-243. These cases are now consolidated as *Webroot, Inc. v. AO Kaspersky Lab*, No. 6:22-cv-00243-ADA-DTG (W.D. Tex.).

2. Administrative Proceedings

There are no other administrative proceedings at issue.

C. Lead and Back-Up Counsel

Petitioners provide the following designation of counsel:

Lead Counsel	Backup Counsel
Ryan Iwahashi (Reg. No. 63,378) Gibson, Dunn & Crutcher LLP 1881 Page Mill Road Palo Alto, CA 94304-1211 Tel: (650) 849-5367 RIwahashi@gibsondunn.com	Josh Krevitt (<i>pro hac vice</i> motion requested) Gibson, Dunn & Crutcher LLP 1881 Page Mill Road Palo Alto, CA 94304-1211 Tel: (212) 351-2490 JKrevitt@gibsondunn.com
	Brian A. Rosenthal (<i>pro hac vice</i> motion requested) Gibson, Dunn & Crutcher LLP 200 Park Avenue New York, NY 10166-0193 Tel: (212) 351-2339 BRosenthal@gibsondunn.com
	Timothy P. Best (<i>pro hac vice</i> motion requested) Gibson, Dunn & Crutcher LLP 333 South Grand Avenue Los Angeles, California 90071-1504 Tel: (213) 229-7659 TBest@gibsondunn.com

Petitioners respectfully request authorization to file motions for Josh Krevitt, Brian Rosenthal, and Timothy Best to appear before the USPTO *pro hac vice*. Messrs. Krevitt, Rosenthal, and Best are experienced litigating attorneys and are currently serving as lead counsel for Petitioners in the District Court Action. Each has established familiarity with the subject matter at issue in this proceeding. Petitioners intend to file a motion to appear *pro hac vice* under 37 C.F.R. § 42.10. Pursuant to 37 C.F.R. § 42.10(b), powers of attorney accompany this petition.

D. Service of Information Under 37 C.F.R. § 42.8(b)(4)

Service of any documents via hand delivery may be made at the mailing address of lead and backup counsel designated above. Petitioners also hereby consent to electronic service, and service via electronic mail may be made at the email addresses provided above for the lead and backup counsel.

III. PAYMENT OF FEES

Pursuant to 37 C.F.R. §§ 42.103 and 42.15(a), the required fee was submitted but the Office is authorized to charge any fee deficiency, or credit overpayment, to Deposit Account 50-1408, including additional fees due in connection with this Petition.

IV. REQUIREMENTS OF *INTER PARTES* REVIEW

Pursuant to 37 C.F.R. § 42.104, each requirement for IPR of the '844 Patent is satisfied.

A. Standing

Pursuant to 37 C.F.R. § 42.104(a), Petitioners certify that the '844 Patent is available for IPR and that Petitioners are not barred or estopped from requesting IPR of the claims on the ground identified herein. Specifically, Petitioners certify that: Petitioners have not filed a civil action challenging the validity of the '844 Patent; this petition is filed not more than one year from March 11, 2022, the date on which Sophos Ltd. was served with the complaint alleging infringement of the '844 Patent;

the estoppel provisions of 35 U.S.C. § 315(e)(1) do not prohibit this *inter partes* review; and this petition is filed after the later of (a) the date that is nine months after the date of the grant of the '844 Patent or (b) the termination of any post-grant review of the '844 Patent.

B. Identification of Challenge and Relief Requested

Pursuant to 37 C.F.R. § 42.104(b), Petitioners request that the Board institute IPR of claims 1–20 of the '844 Patent on one or more grounds under AIA 35 U.S.C. § 103.

The '844 Patent issued from an application filed May 12, 2015. For purposes of this Petition only, Petitioners do not dispute that the Challenged Claims are entitled to an effective filing date of May 12, 2015.

The precise relief requested by Petitioners is that claims 1–20 be canceled based on the grounds below:

Ground	Claims	Basis for Rejection ¹
1	1–20	Obviousness over Ideses
2	1, 8–10, 17	Obviousness over Mori
3	1–20	Obviousness over Ideses in view of Mori

¹ All obviousness grounds include the knowledge of POSITA.

The above challenges are made in view of the following prior art references, none of which was cited or considered during prosecution of the '844 Patent:

1. U.S. Patent Appl. Pub. No. 2015/0213376 (“Ideses”) (Ex. 1005)

Ideses, titled “Methods and Systems for Generating Classifiers for Software Applications,” was filed on January 29, 2015 and published on July 30, 2015. Ideses’s earliest claim to priority on the face of the publication is to a provisional application filed January 30, 2014. Ideses therefore qualifies as prior art under AIA 35 U.S.C. § 102(a)(2). Ideses was neither cited nor considered during prosecution of the '844 Patent.

2. Japanese Patent Appl. Pub. No. 2012027710A (“Mori”) (Ex. 1006)

Mori, titled “Software Detection Methods and Devices and Programs,” was filed on July 23, 2010, and published on February 9, 2012. Mori therefore qualifies as prior art under AIA 35 U.S.C. § 102(a)(1). Mori was neither cited nor considered during prosecution of the '844 Patent.

C. How the Challenged Claims Are to be Construed Under 37 C.F.R. § 42.104(b)(3)

A claim subject to *inter partes* review “shall be construed using the same claim construction standard that would be used to construe a claim in a civil action under 3 U.S.C. § 282(b), including construing the claim in accordance with the ordinary and customary meaning of such claim as understood by one of ordinary

skill in the art and the prosecution history pertaining to the patent.” 37 C.F.R. § 42.100(b) (November 13, 2018); 83 F.R. 51358 (2018). Patent Owner and Sophos Ltd. have submitted proposed constructions for the Challenged Claims in *Webroot Inc., et al. v. AO Kaspersky Lab*, No. 6:22-cv-00243-ADA-DTG (“the Litigation”). Because Petitioners base this Petition on constructions that *Patent Owner* has advocated in the Litigation, the Petition does not present any controversy over claim meaning and the Board need not construe any terms. *Nidec*, 868 F.3d at 1017.

Although Petitioner will maintain that its proposed constructions should be adopted in the Litigation, that fact poses no obstacle to institution. A Petitioner need only identify “[h]ow the challenged claim is to be construed” for purposes of the petition—the Petitioner need not agree that the construction is correct. *10X Genomics, Inc. v. Bio-Rad Labs., Inc.*, IPR2020-00086, Paper 8 at 17–22 (PTAB Apr. 27, 2020) (instituting IPR and concluding that “judicial efficiency will be enhanced by allowing Petitioner to rely upon a claim construction that Patent Owner is relying upon in the related district court litigation to assert infringement”); see also *Western Digital Corp. v. SPEX Techs., Inc.*, IPR2018-00082, Paper 11 at 12 (PTAB Apr. 25, 2018) (“Petitioner’s statement that its Petition ‘is based on the claim constructions urged by Patent Owner’ in related District Court litigation . . . suffices to identify claim constructions . . . in compliance with § 104(b)(3).”).

Petitioners briefly address certain terms below. Petitioners maintain that the Challenged Claims would have been obvious under any reasonable construction.

1. The terms relating to whether “character strings” are “within a predetermined range” (claims 1, 10, and 17)

In the Litigation, Sophos Ltd. has asserted that the terms relating to whether “character strings” are “within a predetermined range” are indefinite. Ex. 1007, 59. Patent Owner maintains that these terms are not indefinite and that no constructions are necessary. Therefore, for purposes of this proceeding, Petitioners propose that no constructions of these terms are necessary, and that the ordinary and customary meaning of these terms at the time of the ’844 Patent apply. *See 10X Genomics, Inc. v. Bio-Rad Labs., Inc.*, IPR2020-00086, Paper 8 at 17-22 (PTAB Apr. 27, 2020).²

² While Sophos Ltd. will maintain in the Litigation that the terms relating to whether “character strings” are “within a predetermined range” are indefinite, the question of indefiniteness is not before the Board and is no barrier to the Board’s inquiry. *Intel Corp. v. Qualcomm Inc.*, 21 F.4th 801, 813 (Fed. Cir. 2021); *Haas Automation, Inc. v. Olati LLC*, IPR2021-00146, Paper 11 at 11–13 (PTAB May 18, 2021) (instituting IPR notwithstanding Petitioner’s position that claim term is subject to Section 112, ¶ 6 and indefinite).

2. “within a predetermined range” (claims 1, 10, and 17)

In the Litigation, Sophos Ltd. has asserted that “within a predetermined range” should be construed as “between a predefined lower and upper limit.” Ex. 1007, 63. Patent Owner disputes that the term is so limited and maintains that no construction of this term is necessary. *See id.* Thus, Petitioners base this Petition on Patent Owner’s claim construction: that no construction of the term “within a predetermined range” is necessary.

3. “support vector processing” (claims 1, 10, and 17)

In the Litigation, Sophos Ltd. has asserted that “support vector processing” should be construed as “processing with a support vector machine.” Ex. 1007, 75. Patent Owner disputes that the term is so limited and maintains that no construction of this term is necessary. *See id.* Thus, Petitioners base this Petition on Patent Owner’s claim construction: that no construction of the term “support vector processing” is necessary.

D. How the Challenged Claims Are Unpatentable Under 37 C.F.R. § 42.104(b)(4)

An explanation of how the Challenged Claims are unpatentable under the statutory grounds identified above, including the identification of where each element of the claim is found in the prior art patents or printed publications, is provided below.

E. Supporting Evidence Under 37 C.F.R. § 42.104(b)(5)

The exhibit numbers of the supporting evidence relied upon and the relevance of the evidence to the challenged claims, including an identification of specific portions of the evidence that support the challenge, are provided below. A list of exhibits identifying the exhibits is also included in this petition. The technical information and grounds for rejection explained in the petition are further supported by the Expert Declaration of Dr. Gene Tsudik, attached as Exhibit 1004.

A List of Exhibits is included in this petition pursuant to 37 C.F.R. § 42.63(e).

V. THRESHOLD REQUIREMENT FOR *INTER PARTES* REVIEW

Under 35 U.S.C. § 314(a), institution of *inter partes* review requires “a reasonable likelihood that the petitioners would prevail with respect to at least one of the claims challenged in the petition.” This petition meets this threshold for each of the asserted ground of unpatentability.

VI. PERSON OF ORDINARY SKILL IN THE ART

A person of ordinary skill in the art (“POSITA”) on May 12, 2015, would have a Bachelor’s degree in Computer Science, Computer Engineering, Electrical Engineering, or an equivalent discipline, and at least one year’s worth of experience relating to static data analysis and malware detection. In the alternative, a POSITA could have two or more years of work experience relating to static data analysis and malware detection in lieu of the education requirements. Ex. 1004, ¶ 16.

VII. STATE OF THE ART

All concepts in this section were well known and widely used by POSITAs at least as of May 12, 2015.

1. Malware

Malware, or malicious software, refers to software developed to modify, steal, or destroy data on a computer or a computer network or to surreptitiously use computer and/or network resources, *e.g.*, crypto-mining malware. See Ex. 1004, ¶ 42. These malicious programs operate in a variety of ways. See *id.*, ¶ 43.

Generally, malware is introduced into a computer or computer network through the Internet or an external storage device. It may be hidden within bundles of software, concealed in a web page, or disguised as a recognizable program. See Ex. 1012, 4–6; Ex. 1004, ¶ 44.

2. Machine Learning / Support Vector Machines

Machine learning (or “ML”) is a sub-field of artificial intelligence (“AI”) that is focused on techniques that “learn” from large amounts of data to improve performance of certain tasks. Ex. 1004 ¶ 45. Within ML, Support Vector Machines (SVMs) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. See *id.*; Ex. 1013, 2. Supervised learning relies on the use of labeled datasets to train classification/prediction algorithms. During a training process, a model “learns” to

evaluate the training dataset to yield the desired outputs. *See* Ex. 1004 ¶ 42; *see generally* Ex. 1013, 2–10.

3. Software Encryption and Packing (Compression)

Since software (initial installation, updates, or patches) can be very large in size, it is often compressed to minimize its transportation size. Packing (or compression) aims to minimize the size of software by reduction or elimination of unnecessary or redundant components or information. Ex. 1004 ¶ 46. Packing/compression is thus a form of obfuscation, as compressed software must be decompressed before the underlying object code can be analyzed or executed. *See* Ex. 1010, 11–12; Ex. 1012, 9.

Software may be encrypted in order to make it more difficult for a third party to reverse engineer, decompile, or disassemble the executable. *Id.* ¶ 47. Encrypted software must be decrypted before the underlying object code can be analyzed or executed. When software is a paid and/or restricted commodity, it may be encrypted after compression such that only authorized recipients can decrypt it. *See id.*

4. Static and Dynamic Malware Analysis

Static analysis is the (typically automated) analysis of a target computer program that is performed without executing it. This is in contrast to dynamic analysis, which is performed upon a target program during its (usually sandboxed) execution. Ex. 1004, ¶ 49; *see* Ex. 1010, 6–7; Ex. 1012, 7–8.

VIII. OVERVIEW OF THE '844 PATENT

The '844 Patent purports to describe a method for detecting “whether an executable file is harmful or benign before the executable file is actually executed on a processing device.” Ex. 1001, 2:4–6. More specifically, the '844 Patent is directed to a machine-learning technique called a “support vector machine” to detect malware. *Id.*, 7:46–48, 9:41–67. The '844 Patent teaches that a “plurality of static data points may be extracted from an executable file.” *Id.*, 1:21–23. More specifically, these features may be extracted without decrypting or unpacking the executable file. *Id.*, 1:21–23; 2:11–14; *see also id.*, 12:10–19 (describing analyzing an executable file “without decrypting or unpacking” it), 12:36–38 (same). The '844 patent describes generating a “feature vector” from these extracted features. *Id.*, 8:35–48. The '844 Patent further explains that the number of features collected and analyzed for an executable file can vary, and that the alleged invention “can adaptively set features” of the feature vector “based on the static data points extracted for a file.” *Id.*, 7:57–59; *see id.*, 6:60–19. For example, “one or more features of the feature vector may be selectively turned on or off based on evaluation of whether a value of a static data point is within a predetermined range.” *Id.*, 12:60–63. Once the feature vector is generated, the Challenged Claims require using “support vector machine[s] (SVM)” which may perform “data analysis and pattern

recognition” on the feature vector to determine whether the file is malicious. *Id.*, 9:41–59.

A. Summary of the Prosecution History of the ’844 Patent

The application that issued as the ’844 Patent, U.S. application no. 14/709,875 (“the ’875 application”), was filed on May 12, 2015. Ex. 1001, Cover. As filed, the ’875 application contained 20 claims, of which claims 1, 10, and 19 were independent. Ex. 1002, 49–52.

On December 8, 2016, the examiner issued a nonfinal rejection, finding pending claims 1, 2, 10, 11, and 19 anticipated by U.S. Patent No. 9,465,940³ to Wojnowicz *et al.* *Id.*, 92–97. The examiner also rejected the remaining claims as obvious over Wojnowicz in view of additional references. *Id.*, 97–107.

In response, the applicant filed amendments to independent claims 1, 10, and 19, each of which further limited the “extracting a plurality of static data points” step to “wherein the static data points correspond to one or more points of reference used to evaluate the executable file.” *Id.*, 127–32. On March 24, 2017, the examiner issued a final rejection on the same grounds as the previous rejection. After

³ The patent number of the Wojnowicz reference was incorrectly identified in the office actions as “8,709,924,” but was correctly identified in the PTO-892 (Notice of References Cited) as “US-9,465,940.” *Compare* Ex. 1002, 94, with *id.*, 109.

additional prosecution, including a request for continued examination, further amendments and rejections, and an interview, the applicant filed a second request for continued examination in which it: (1) removed the “points of reference” limitations previously added; (2) limited the “extracting a plurality of static data points step” to “wherein the plurality of static data points represent predefined character strings in the executable file;” and (3) made further amendments, including to require “a classifier . . . trained to classify the plurality of static data points based on a collection of data . . . wherein the collection of data comprises the plurality of static data points.” *Id.*, 304–09.

On July 17, 2019, the examiner issued a nonfinal rejection over a new set of prior art references, specifically finding pending claims 1–2, 10–11, and 19 anticipated by “Detection of Packed Malware” by Devi *et al.* *Id.*, 320–23. The examiner likewise rejected claims 3-4, 6-9, 12-13, 15-18, and 20 obvious over Devi in combination with various references. *See id.*, 323–27. The examiner noted, however, that claims 5 and 14 would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims. *Id.*, 327. Pending dependent claims 5 and 14 recited “wherein features of the feature vector are selectively turned on or off based on evaluation of whether a value of a static data point from the plurality of extracted static data points is within a predetermined range.” *Id.*, 305, 308.

Following interviews on September 23, 2019, and November 1, 2019, the applicant filed amendments to the independent claims “to include the allowable subject matter of previously pending claims 5 and 14,” *id.*, 368–72, and agreed to further examiner amendments that would incorporate “some of the intervening claim language of claims 3 and 4.” *Id.*, 411–12. A notice of allowability was issued on January 24, 2020, and the ’844 Patent issued on March 24, 2020. *Id.*, 424–42, 456.

IX. DETAILED CHALLENGE

A. Ground 1: Claims 1–20 Are Rendered Obvious By Ideses

As described below, Ideses teaches each and every element of the Challenged Claims.

1. Prior Art Overview

Ideses discloses “a method for training a classifier for classifying applications.” Ex. 1005, Abstract. Specifically, Ideses discloses a method for providing “run-time classification of software applications, . . . for example, to detect if the software application for installation [on a mobile device] is malware, adware, or benign (e.g., fine for installation).” *Id.*, [0123]. Although Ideses generally describes *run-time* feature extraction (*i.e.*, during execution of a software application), such run-time extraction is noted as an *optional* feature, and Ideses contemplates extracted features that “may contain static meta data regarding the application, for example, icon, name, rating or other features,” as well as

“information regarding the executable code and/or software package, for example, in the form of byte code, resources, permissions, or other features.” *Id.*, [0089]; see *id.*, [0133] (“Optionally, features are extracted during run-time.”). Ideses discloses that “the extracted features are collated into feature vectors,” and that the subject “software application is classified . . . by a run-time classification module,” which can label the software application, for example, “as benign or adware.” *Id.*, [0134]–[0135].

2. Analysis

a. Independent Claim 1

[1.pre]: A computer-implemented method comprising:

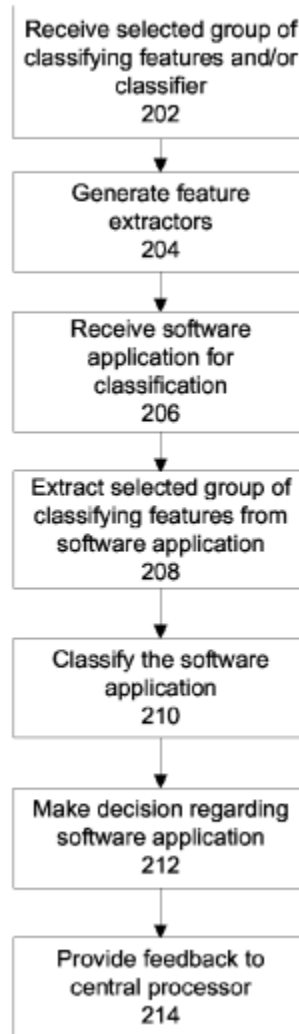
To the extent the preamble of claim 1 is limiting, Ideses discloses [1.pre]. Ex. 1004 ¶¶ 60–62. Ideses discloses “methods and systems for generating classifiers for software applications.” Ex. 1005, [0002]. Ideses further discloses “computer readable program instructions . . . for causing a processor to carry out aspects of the present invention.” *Id.*, [0063]; see also *id.*, [0064]–[0069].

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

This limitation includes multiple parts, each of which is disclosed by Ideses. Ex. 1004 ¶ 63.

Extracting a plurality of static data points from an executable file. Ideses discloses this portion of element [1.a]. Ex. 1004 ¶¶ 64–67. Figure 2 of Ideses “is a

flowchart of a method of run-time execution of a classifier based on a pruned feature list”:



Ex. 1005, [0123], Fig. 2. At step 206, “a software application,” *i.e.*, an “executable file” as required by element [1.a], “is received at the mobile device,” for example, having been “downloaded from the internet.” *Id.*, [0131]. At step 208, “features,” *i.e.*, “a plurality of static data points” as required by element [1.a], “are extracted based on the software application.” *Id.*, [0132]; Ex. 1004 ¶ 65.

The '844 Patent broadly describes “static data points” as “point[s] of reference used to evaluate an executable file.” Ex. 1001, 5:62–63. The '844 Patent explains that “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.” *Id.*, 5:63–6:3; *see also* Ex. 1002, 312 (distinguishing “fixed-length file chunks” that are “random . . . character string[s] having no initial significance” from the claimed “static data points [that] represent one or more predefined character strings”).

Ideses discloses the extraction of such “static data points.” Ex. 1004 ¶ 67. Specifically, Ideses explains that “the extracted features may contain *static meta data* regarding the application, for example, icon, name, rating or other features,” and may further contain “information regarding the executable code and/or software package, for example, in the form of byte code resources, permissions, or other features.” Ex. 1005, [0089].⁴ The “static meta data” described in Ideses corresponds to the “static data points” required by element [1.a], and the examples listed in Ideses match examples of “static data points” identified in the '844 Patent. For instance:

⁴ Emphasis added unless otherwise noted.

Ideses	'844 Patent
“icon”	“resource information”
“name”	“string . . . information,” “company name,” “internal name,” “product name”
“information regarding the executable code and/or software package . . . [e.g.,] byte code, resources”	“resource information,” “program information (e.g., APIs and DLLs),” “package code”
“permissions”	“file flags”

See *id.*; Ex. 1001, 5:63–6:1, Table 1.1; Ex. 1004, ¶ 67.

Without decrypting or unpacking the executable file. Ideses discloses this portion of element [1.a]. Ex. 1004 ¶¶ 68–70. Regarding the extraction step shown in Ideses’ Figure 2, Ideses explains that “[o]ptionally, features are extracted during run-time,” Ex. 1005, [0133], and Ideses also discloses conversely that “[t]he feature extraction may take place off-line, not part of a run-time operation.” *Id.*, [0090]. While run-time feature extraction would require decryption or unpacking the executable file, off-line feature extraction would be conducted without decrypting or unpacking the executable file. See Ex. 1004 ¶ 70.

Ideses explains that “[t]he extracted features may be varied, containing information from different modalities,” and distinguishes between “static meta data” (which can be collected “off-line”) and “behavioral features, for example, temporal

information regarding system calls, system usage, CPU usage, [and] network utilization,” which can only be extracted at “run-time,” (during which the executable file is necessarily decrypted and/or unpacked). *See* Ex. 1004, ¶ 68.

Ideses lists a number of features that may be extracted, including “*static meta data* regarding the application, for example, icon, name, rating[,] . . . information regarding the executable code and/or software package, for example, in the form of byte code, resources, permissions, or other features.” Ex. 1005, [0089]. It would have been obvious to a POSITA reading Ideses to perform “off-line” extraction of the “static meta data” without attempting to first decrypt or unpack the executable file. *See* Ex. 1004, ¶¶ 69–70.

Indeed, it was well known prior to the effective filing date of the ’844 Patent that a “basic” static malware analysis technique was to search through a file’s strings “to get hints about the functionality of a program,” and that “[m]alware that is packed or obfuscated contains very few strings.” *See* Ex. 1010, 8–10; Ex. 1004, ¶ 70; *see also, e.g.*, Ex. 1010, 12 (explaining that an “unpacking stub can be viewed by a malware analyst” and that “[i]f [a malware analyst] attempt[s] to perform static analysis on the packed program, [the analyst] will be analyzing the stub, not the original [malware] program”). Thus, it would have been obvious to a POSITA that extracting “static meta data” from a potential malware executable without first

unpacking or decrypting the executable would have been the first step in a basic static analysis. *See* Ex. 1004, ¶ 70.

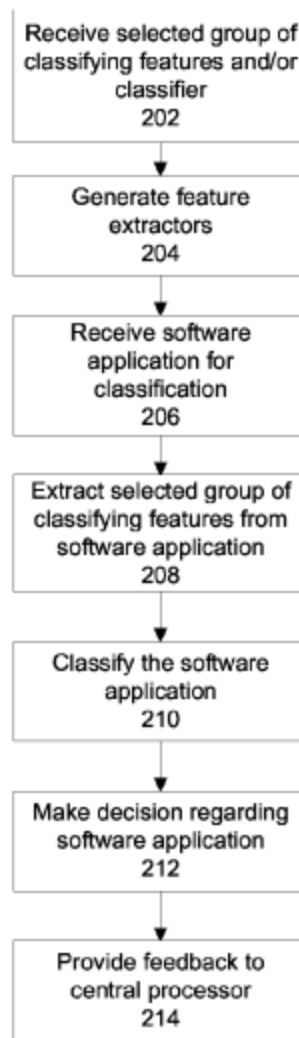
Wherein the plurality of static data points represent predefined character strings in the executable file. Ideses discloses this portion of element [1.a]. Ex. 1004, ¶¶ 71–72. Ideses explains that “[t]he extracted features *may be any feature that describes the software application*,” including “static meta data regarding the application, for example, *icon, name, rating or other features*,” and the extracted features “may contain information regarding the executable code and/or software package, for example, in the form of byte code, resources, permissions, or other features.” Ex. 1005, [0089]. A POSITA would have understood, or it would have been obvious from Ideses that “any feature that describes the software application” would include “predefined character strings” in the executable file. *See* Ex. 1004, ¶ 72. Several of the example features listed in Ideses, such as the application “name” and “information regarding . . . the software package” would comprise “predefined character strings.” *See id.*

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

This limitation includes multiple parts, each of which is disclosed by Ideses.

Ex. 1004 ¶ 73.

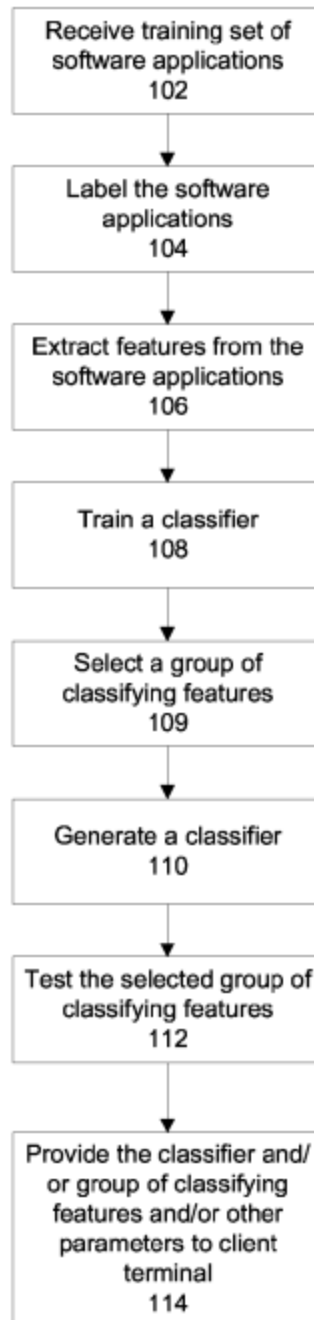
Generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points: Ideses discloses this portion of element [1.b]. Ex. 1004, ¶¶ 74–75. As explained above with respect to element [1.a], Figure 2 of Ideses “is a flowchart of a method of run-time execution of a classifier based on a pruned feature list”:



Ex. 1005, [0123], Fig. 2. At step 202, “*the trained classifier . . . is received by the mobile device.*” *Id.*, [0124]. Ideses explains that the “features extracted” at step 208 of Figure 2 (*i.e.*, “the plurality of static data points” required by element [1.a]), “are based on . . . *the trained classifier* received from the central server” in block 202. *Id.*, [0132]. Ideses further explains that “the extracted features are collated into *feature vectors.*” *Id.*, [0134]. Ideses therefore discloses “generating a feature vector from the plurality of static data points using a classifier trained to classify the

plurality of static data points,” as required by this portion of element [1.b]. Ex. 1004, ¶ 75.

Based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files: Ideses discloses this portion of element [1.b]. Ex. 1004, ¶¶ 76–79. Figure 1 of Ideses depicts “a method of generating a selected group of classifying features (e.g., feature vector) for classification of software applications,” *i.e.*, a “classifier” as claimed:



Ex. 1005, [0071], Fig. 1. At step 102, “multiple software applications for training the classifier are received,” which may be “manually provided by an operator of the system (e.g., the manufacturer), provided by software application manufacturers,

automatically downloaded from the internet, provided by updates from mobile devices, . . . and/or provided by other methods.” *Id.*, [0080].

At step 104, “the multiple software applications are labeled with a classification type based on desired software application classification categories,” which “denote *desired* or *undesired* software applications.” *Id.*, [0081]. Ideses also mentions “Adware, Goodware, Intrusive Adware,” as other classification types that may be used. *Id.*, [0083]; *see also id.*, [0141] (“[T]he software device is installed on the mobile device, for example, if the classification type is determined to be benign, or other non-harmful and/or beneficial classification types. Alternatively, the software application is not installed, deleted, flagged, or otherwise prevented from functioning on the mobile device, for example, if the classification type is determined to be harmful, malware, bothersome, adware, intrusive, spyware, or other non-desirable classification types.”). These classification types correspond to the “known malicious executable files” (*i.e.*, “harmful” or “malware”) and “known unwanted executable files” (*i.e.*, “undesired,” such as “Adware,” “Intrusive Adware,” “bothersome,” or “spyware”) and “known benign executable files” (“desired,” “benign, or other non-harmful and/or beneficial classification types”). *See* Ex. 1004, ¶ 78.

At step 106, “features are extracted from the software applications,” including, for example, “data and/or meta data.” *Id.*, [0086]–[0089]. And, at step

108, “a classifier is trained based on the set of extracted features (block 106) and software classification labeling (block 104).” *Id.*, [0094]. Ideses therefore discloses “generating a feature vector . . . based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files,” as required by this portion of element [1.b]. Ex. 1004, ¶ 79.

Wherein the collection of data comprises at least a portion of the plurality of static data points. Ideses discloses this portion of element [1.b]. Ex. 1004, ¶¶ 80–81. As explained above, the features that are extracted from the software applications at step 106 of Figure 2 “may be any feature that describes the software application,” including “static meta data regarding the application, for example, icon, name, rating or other features,” as well as “information regarding the executable code and/or software package, for example, in the form of byte code, resources, permissions, or other features.” Ex. 1005, [0086]–[0089]. These features can be “stored as feature vectors, which may be used for training the classifier,” as depicted in step 108. *Id.*, [0091], [0094]. And, regarding step 204 of Figure 2 (“Generate feature extractors”), Ideses indicates that “the feature extractor modules are automatically built and coded *based on the pruned features* that have been selected by the central processor.” *Id.*, [0127]; *see id.*, [0132] (“Optionally, the features extracted are based on the group of classifying features and/or trained classifier received from the central server (block 202).”). Ideses therefore discloses

“generating a feature vector . . . wherein the collection of data comprises at least a portion of the plurality of static data points,” as required by this portion of element [1.b]. Ex. 1004, ¶ 81.

Wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range. Ideses discloses this portion of element [1.b]. Ex. 1004, ¶¶ 82–84. Ideses explains that in the method depicted in Figure 1, “[t]he full feature list is optionally selectively pruned,” *i.e.*, “selectively turned off” as required by element [1.b], “to generate a pruned feature list.” Ex. 1005, [0071]. Further, “each feature of the group is selected from-the-ground-up; added to the group as opposed to removed from or retained within a larger group,” *i.e.*, “selectively turned on” as required by element [1.b]. *Id.*

Ideses explains that “[t]he group of classifying features may be selected based on one or more methods,” and that “the group of classifying features is selected to reduce the dimensionality of the feature vector.” *Id.*, [0104]–[0105]. For example, the group of classifying features may be selected “based on the significance of the extracted features,” such that significant features, or “those that have a large effect on the classification outcomes, may be retained,” whereas insignificant features, or “those that have a small, negligible or no effect on classification outcome may be pruned.” *Id.*, [0106].

Ideses provides two approaches to pruning insignificant features. First, “features that correspond to coefficients having a value of zero are pruned.” *Id.*, [0107]. Second, “features with a low magnitude value of the coefficient are identified for pruning and/or are not included.” *Id.* Ideses explains that selection of the group of classifying features based on coefficients with zero/low values may be denoted by the following expressions, respectively:

$$\text{Feature}_i \in FVf | (\text{Coefficient}_i) \neq 0$$

$$\text{Feature}_i \in FVf | (\text{Coefficient}_i) \geq \alpha$$

Id., [0109]–[0110]. Both of these approaches to selectively including/excluding features are based on whether a feature coefficient “is within a predetermined range” as required by this portion of element [1.b]. Moreover, Ideses discloses that just such an approach can be used to generate a “pruned feature vector” (*i.e.*, a vector in which features of the feature vector are selectively turned on or off) for classification by a classifier. *Id.*, [0157], Fig. 5; Ex. 1004, ¶ 84. Ideses therefore discloses “generating a feature vector . . . wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range,” as required by this portion of element [1.b]. Ex. 1004, ¶ 84.

[1.c]: evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

Ideses discloses [1.c]. Ex. 1004, ¶¶ 85–88. At step 210 of Ideses’ Figure 2, “the software application is classified,” for example, “as benign or adware.” Ex. 1005, [0135]. Ideses explains that “the software application is classified by applying the training classifier received from the central server,” and that alternatively or additionally, “the software application is classified based on the group of classifying features that have been computed by the feature extractor” at step 208. *Id.*, [0137].

Ideses explains that the classifier may be “trained based on supervised learning,” an example of which are “Support Vector Machines.” *Id.*, [0096]; *see also id.*, [0107] (“[F]eatures that correspond to coefficients having a value of zero are pruned or not included, for example, when classification is performed *based on a support vector machine* (SVM).”). A POSITA would have understood Ideses’ disclosure of classification “based on a support vector machine” to be synonymous with “using support vector processing” as required by element [1.c]. *See* Ex. 1004, ¶ 88.

b. Claim 2

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

Ideses renders obvious this additional element. Ex. 1004, ¶¶ 89–93. Ideses discloses that “[t]he extracted features may be any feature that describes the software application,” including “static meta data regarding the application, for example, icon, name, rating or other features.” Ex. 1005, [0089]. And, “the extracted features may contain information regarding the executable code and/or software package, for example, in the form of byte code, resources, permissions, or other features.” *Id.* It would have been obvious to one of skill in the art to “classify[] the extracted static data points according to a type of data extracted from the executable file” by storing (*i.e.*, “encoding” as required by claim [2]) the extracted features in memory using a data structure or primitive data type that corresponds to the type of data extracted. Ex. 1004, ¶ 93. These features enumerated in Ideses correspond to varying types of data, for example:

Ideses	'844 Patent
“icon,” “byte code,” “resources”	“byte sequences”
“name”	“string sequences”
“rating”	“numeric values”
“permissions”	“nominal values”

Id., ¶ 92. Byte sequences (such as an application icon or byte code) would have been stored/encoded using byte arrays, string sequences (such as an application name) would have been stored in a customary format such as C-style strings (arrays of characters having a null terminator), and numeric values (such as an application rating) would have been stored using a primitive integer data type. *Id.*, ¶ 93. Thus, the extracted features (*i.e.*, the “static data points”) would have been “classif[ied]” based on “the type of data extracted” and encoded by virtue of their storage in memory in an appropriate data type. *Id.*

c. Claim 3

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

Ideses discloses this additional element. Ex. 1004, ¶¶ 94–98. As explained above with respect to element [1.b], Ideses discloses that a client device can generate a “pruned feature vector” (*i.e.*, a vector in which features of the feature vector are selectively turned on or off) for classification by a classifier. Ex. 1005, [0157], Fig. 5; Ex. 1004, ¶ 95. Ideses explains that in the method depicted in Figure 1, “[t]he full feature list is optionally selectively pruned,” *i.e.*, “selectively turned off” as required by element [1.b], “to generate a pruned feature list.” Ex. 1005, [0071]. Further, “each feature of the group is selected from-the-ground-up; added to the group as

opposed to removed from or retained within a larger group,” *i.e.*, “selectively turned on” as required by element [1.b]. *Id.*

Ideses also discloses “selectively setting features of the classifier *based on the plurality of extracted static data points*,” as required by claim [3]. Ex. 1004, ¶ 96. Specifically, Ideses discloses selecting “a group of classifying features” with the goal of “reducing client terminal processor usage and/or client terminal memory requirements.” Ex. 1005, [0016]. Ideses explains that “a certain feature may have strong classification effectiveness and/or discrimination capability, but may place large requirements in terms of memory and/or CPU usage,” whereas a “different feature may have slightly less classification effectiveness and/or discrimination capability, but significantly lower CPU and/or memory usage.” *Id.*, [0055]. In order to reduce the processor usage and/or the memory requirements of the mobile device, the lower CPU/memory feature may be selected over the feature having larger requirements. *Id.*, [0053]–[0055].

The use of byte sequence-type features (such as byte code or application resources) may require more computational cost than numeric-type features (such as a rating). Ex. 1004 ¶ 98. Thus, it would have been obvious to a POSITA reading Ideses to selectively set features of a classifier (*e.g.*, by not setting byte code features but setting numeric features) based on the type of data extracted (*i.e.*, the “classification” of the extracted features) in order to reduce usage of limited

resources available to the client. *Id.*; see Ex. 1005, [0155]–[0157] (“[C]lient . . . is a resource-limited device, having limited CPU availability, limited CPU power, and/or limited memory. . . . The [pruned] set of extracted features *is selected to take place during run-time*, under the available resources of [the] client.”).

d. Claim 4

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

Ideses renders obvious this additional element. Ex. 1004, ¶¶ 99–102. As explained above with respect to claim [2], Ideses discloses that “[t]he extracted features may be any feature that describes the software application,” including “static meta data regarding the application, for example, icon, name, rating or other features.” Ex. 1005, [0089]. And, “the extracted features may contain information regarding the executable code and/or software package, for example, in the form of byte code, resources, permissions, or other features.” *Id.* It would have been obvious to one of skill in the art to “classify[] the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values” by storing the extracted features in memory using a data structure or primitive data type that corresponds to the type of data extracted. Ex. 1004, ¶ 101. These features correspond to varying types of data, for example:

Ideses	'844 Patent
“icon,” “byte code,” “resources”	“byte sequences”
“name”	“string sequences”
“rating”	“numeric values”
“permissions”	“nominal values”

See *id.*, ¶ 92. And, as explained above, byte sequences would have been stored using byte arrays, string sequences would have been stored as conventional C-style strings, and numeric values would have been stored using a primitive integer data type. *Id.*, ¶ 101.

It would have further been obvious to one of skill in the art to classify the extracted features of Ideses into additional categories that correspond to other data types, such as “nominal” and Boolean values, as required by claim [4]. *Id.*, ¶ 102. For example, permissions would have been stored as bit fields, with each bit set to ‘1’ representing a granted permission. Likewise, Boolean values (which have one of two possible values, “true” or “false”) would have been stored using a single bit or as a primitive integer type (with ‘0’ representing false, and all other values representing “true”). *Id.*

e. Claim 5

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

Ideses discloses this additional element. Ex. 1004, ¶¶ 103–05. As explained above with respect to element [1.b], Ideses discloses that “the extracted features are collated into feature vectors” as required by claim [5]. Ex. 1005, [0134]. Further, and as also explained above with respect to element [1.b], Ideses provides two approaches to pruning insignificant features (*i.e.*, features that have a “small, negligible or no effect on classification outcome”) while retaining significant features (*i.e.*, features “that have a large effect on the classification outcomes”). First, “features that correspond to coefficients having a value of zero are pruned.” *Id.*, [0107]. Second, “features with a low magnitude value of the coefficient are identified for pruning and/or are not included.” *Id.* Ideses explains that selection of the group of classifying features based on coefficients with zero/low values may be denoted by the following expressions, respectively:

$$\text{Feature}_i \in FV \text{ if } |(\text{Coefficient}_i)| \neq 0$$

$$\text{Feature}_i \in FV \text{ if } |(\text{Coefficient}_i)| \geq \alpha$$

Id., [0109]–[0110]. These feature coefficients described by Ideses are “weights” that indicate the significance of a given feature. Ex. 1004, ¶ 105. And, as explained

above with respect to element [1.b], Ideses discloses that a client device can generate a “pruned feature vector” (*i.e.*, a vector in which features of the feature vector are selectively turned on or off) for classification by a classifier. *Id.*, [0157], Fig. 5; Ex. 1004, ¶ 104. Thus, a POSITA would have understood that the “pruned feature vector” generated by the client would have included features “weighted” based on the coefficient values described above. *See* Ex. 1004, ¶ 105.

f. Claim 6

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

Ideses discloses this additional element. Ex. 1004, ¶¶ 106–07. Ideses discloses that at step 210 of Figure 2, “a certainty of the classification [of the software application] is provided, for example, *the estimated probability that the classification is correct.*” Ex. 1005, [0140]. Ideses explains that the “decision to install or remove the software application may be based on the certainty level,” and that “high probability malicious software may be automatically (or manually) removed.” *Id.*, [0143]. Removal of the “high probability malicious software” corresponds to “preventing execution of the executable file” as required by claim [6]. Ex. 1004, ¶ 107.

g. Claim 7

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

Ideses discloses this additional element. Ex. 1004 ¶¶ 108–11. See claim [6], above. As noted above, Ideses discloses that “a certainty of the classification is provided, for example, *the estimated probability that the classification is correct.*” Ex. 1005, [0140]. As explained by Petitioners’ expert, “the estimated probability that the classification is correct” as used in Ideses refers to a measure of *how accurate* the classifier actually is. Ex. 1004, ¶ 110. One commonly-used measure of classifier accuracy is the *false positive rate*, or “FPR,” which is the proportion of negative cases (*i.e.*, benign files) that are incorrectly identified as positive cases (*i.e.*, malware). In other words, the FPR is the probability that a false positive will be raised. *Id.* FPR is defined as the total number of negative cases incorrectly identified as positive cases (*i.e.*, false positives) divided by the total number of actual negative cases (*i.e.*, false positives plus true negatives). *Id.* Accordingly, it would have been obvious to one of skill in the art reading Ideses that “the estimated probability that the classification is correct” would have been based on the commonly-used FPR, which is “predetermined false positive range data” as required by claim [7]. Ex. 1004, ¶ 110–11.

h. Claim 8

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

Ideses discloses this additional element. Ex. 1004, ¶¶ 112–13. Ideses explains that “a learning module . . . of a *machine learning algorithm* is applied to train the classifier.” Ex. 1005, [0094]; *see id.*, [0095] (“[T]he learning module performs the *machine learning* and/or classifier training.”).

i. Claim 9

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

Ideses discloses this additional element. Ex. 1004, ¶¶ 114–16. Ideses explains that following classification (step 210 of Figure 2), at step 214, “the mobile device reports back to the server,” and “the classification type outcome” (*i.e.*, “harmful, malware, bothersome, adware, intrusive, spyware,” etc.) is reported. Ex. 1005, [0144]. Ideses explains that “the central server learns about the existence of new software applications based on the feedback provided from the mobile device,” and that “the central server re-labels or confirms the labeling of existing software applications based on the provided feedback.” *Id.*, [0145]–[0146]. As an example, “classification results with high certainty received from multiple mobile devices may cause the central server to change the existing classification type, or to retain the existing classification type.” *Id.*, [0146].

j. Independent Claim 10

[10.pre]: A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:

See element [1.pre], above.

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

See element [1.a], above.

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

See element [1.b], above.

[10.c]: evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

See element [1.c], above.

k. Claim 11

[11]: The computer-readable storage device according to claim 10, wherein the extracting of the plurality of static data points executed by the processor further comprises classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classifying.

See claim [2], above.

l. Claim 12

[12]: The computer-readable storage device according to claim 11, wherein generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.

See claim [3], above.

m. Claim 13

[13]: The computer-readable storage device according to claim 11, wherein the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.

See claim [4], above.

n. Claim 14

[14]: The computer-readable storage device according to claim 13, wherein analyzing the executable file further comprises generating a feature vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.

See claim [5], above.

o. Claim 15

[15]: The computer-readable storage device according to claim 10, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.

See claim [6], above.

p. Claim 16

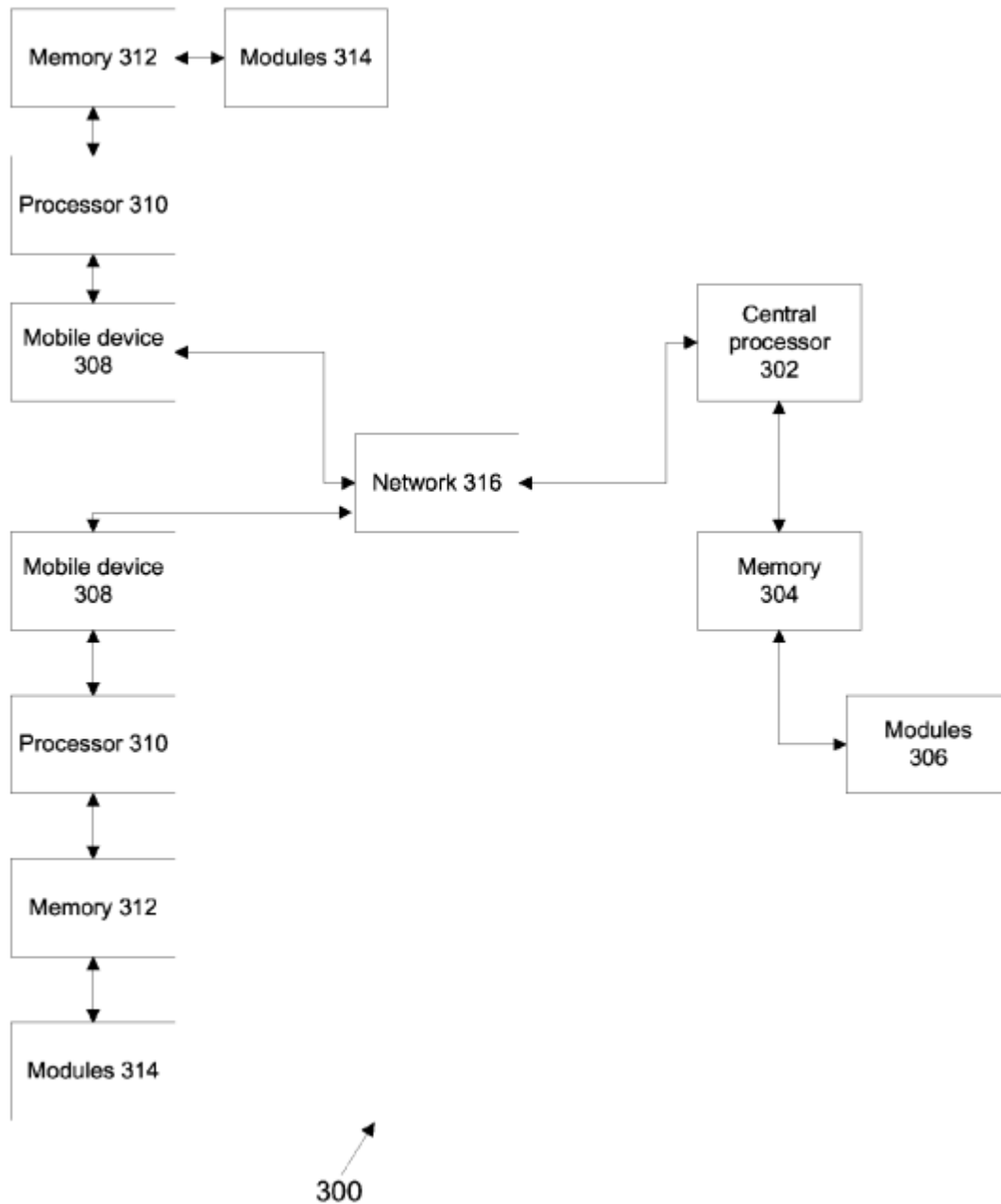
[16]: The computer-readable storage device according to claim 15, wherein the threshold value is set based on predetermined false positive range data.

See claim [7], above.

q. Claim 17

[17.pre]: A system comprising: at least one memory; and at least one processor operatively connected with the memory and configured to perform operation of:

To the extent the preamble of claim 17 is limiting, Ideses discloses [17.pre]. Ex. 1004 ¶¶ 128–30. Ideses discloses “methods and systems for generating classifiers for software applications.” Ex. 1005, Abstract. Specifically, Ideses discloses “a system 300 for generating a selected group of classifying features for classification, and for classifying based on the selected group.” Ex. 1005, [0072].



Id., Fig. 3. Ideses explains that “[s]ystem 300 includes a central processor 302 with a memory 304 for storing data modules 306 thereon.” *Id.*, [0073]; *see also id.*, [0075] (“Mobile devices 308 may be resource limited, having smaller and/or less

powerful processor 310, and/or smaller and/or slower memory 312. Modules 314 are stored on memory 312 for execution by processor 310.”).

[17.a]: extracting a plurality of predefined character strings from an executable file without decrypting or unpacking the executable file;

See element [1.a], above.

[17.b]: generating a feature vector from the plurality of predefined character strings using a classifier trained to classify the plurality of predefined character strings based on a collection of data comprising known malicious executable files, known benign executable files and known unwanted executable files, wherein the collection of data comprises at least a portion of one or more of the plurality of predefined character strings, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more predefined character strings from the plurality of predefined character strings is within a predetermined range; and

See element [1.b], above.

[17.c]: evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

See element [1.c], above.

r. Claim 18

[18]: The system according to claim 17, wherein the determining further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value, and wherein the threshold value is set based on predetermined false positive range data.

See claims [6] and [7], above.

s. Claim 19

[19]: The system according to claim 17, wherein the extracting the plurality of predefined character strings further comprises classifying the extracted plurality of predefined character strings according to a type of data extracted from the executable file, and encoding the extracted plurality of predefined character strings for the classifier based on the classification.

See claim [2], above.

t. Claim 20

[19]: The system according to claim 17, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.

See claim [6], above.

B. Ground 2: Claims 1, 8–10, and 17 Are Rendered Obvious By Mori

As described below, Mori teaches each and every element of claim 1, 8–10, and 17.

1. Prior Art Overview

Mori describes “a software detection device that detects malicious software (hereinafter referred to as ‘malware’).” Ex. 1006, [0010]. Mori explains that “[m]alware often uses deliberately complex encryption and compression (called packing) to avoid detection by pattern matching or static analysis by inverse assemblers,” and that, in general, “in order to perform a detailed analysis of malware, packed executable files need to be decrypted using various techniques.” *Id.*, [0003]–[0004]. Mori notes, though, that methods “that use decryption, disassembly, and

debuggers are time-consuming and require computational resources, and so are not suitable for use in detecting communications containing malware in real time.” *Id.*, [0007].

To solve this problem, Mori “focuses on readable strings contained in software files, collects information (characteristic vectors) that characterizes malware, such as messages contained in programs that constitute malware, host names, execution command names, function names and library names used for communication,” and “compares this content with strings contained in files being inspected to detect malware.” *Id.*, [0023]. In particular, Mori discloses a malware detection method that includes a “a word extraction means that extracts as a word a string of printed and readable bytes ” from a potential malware object, a “a feature extraction step wherein the feature extraction means configures feature vectors for each [potential malware] Object based on a set of words extracted by the word extraction step,” and a determination as to “whether or not the Object is malware based on newly observed feature vectors by applying a machine learning system to the feature vector.” *Id.*, [0016].

2. Analysis

a. Independent Claim 1

[1.pre]: A computer-implemented method comprising:

To the extent the preamble of claim 1 is limiting, Mori discloses [1.pre]. Ex. 1004, ¶¶ 143–45. Mori discloses “a software detection method, apparatus, and program for detecting malware or communications transmitting and/or receiving malware using supervised machine learning systems.” Ex. 1006, [0001]. And, “[t]he operation of each component of the detection device shown in FIG. 1 above can be programmatically constructed and then installed and executed on a computer utilized as the detection device.” *Id.*, [0067].

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

This limitation includes multiple parts, each of which is disclosed by Mori. Ex. 1004, ¶ 146.

Extracting a plurality of static data points from an executable file. Mori discloses this portion of element [1.a]. *Id.* ¶ 147. In the first step of Mori’s malware detection method, “the file or communication data to be sent or received (hereinafter, the file or communication data is collectively referred to as the ‘Object’) is recorded by the Object recording unit” Ex. 1006, [0028]. Next, “a word extraction unit . . . extracts the printed and readable byte strings as words from the Object recorded above, and uses it as a word set.” *Id.*, [0029]; *see also id.*, [0025] (“Readable strings

include messages contained in programs that make up malware, host names used for communication, execution command names, function names, library names, and so on, and can collect a variety of information about malware.”). Mori explains that its method takes advantage of the fact that “[a]lthough the majority of malware today is known to develop a large number of variants after the original is developed,” that *“it is likely that the characteristics of the printable strings [in a variant] will be similar to the original.”* *Id.*, [0026].

Without decrypting or unpacking the executable file. Mori’s malware detection method is performed “without decrypting or unpacking the executable file” as required by element [1.a]. See Ex. 1004, ¶¶ 148–49. Mori explains that “[m]alware often uses deliberately complex encryption and compression (called packing) to avoid detection by pattern matching or static analysis by inverse assemblers,” and that “[p]acked executable files decrypt themselves at runtime and are then deployed in memory.” Ex. 1006, [0003]. And, in general, “in order to perform a detailed analysis of malware, packed executable files need to be decrypted.” *Id.*, [0004]. Mori’s method is an effort to detect malware without using “decryption, disassembly, and debuggers,” as such techniques “are not suitable for use in detecting communications containing malware in real time.” *Id.*, [0007]. Accordingly, Mori focuses on the analysis of “readable strings contained in software

files,” which “can collect a variety of information about malware.” *Id.*, [0023], [0025]; Ex. 1004, ¶ 149.

Wherein the plurality of static data points represent predefined character strings in the executable file. Mori discloses this portion of element [1.a]. Ex. 1004 ¶ 150. Mori explicitly states that its method “focuses on readable strings contained in software files,” which “include messages contained in programs that make up malware, host names used for communication, execution command names, function names, library names, and so on, and can collect a variety of information about malware.” Ex. 1006, [0023], [0025]. Mori explains that its method is useful for detecting malware “variants where only a part of the code has been modified,” because “*it is likely that the characteristics of the printable strings [in a variant] will be similar to the original.*” *Id.*, [0026]. Accordingly, Mori’s word extracting unit uses a “word dictionary . . . prepared in advance” and “extract[s] *only those words that match the words in word dictionary.*” *Id.*, [0050]. In other words, Mori discloses that the extracted words “represent predefined character strings in the executable” to the extent that they match the words in a word dictionary “prepared in advance,” that is, “predefined.” See Ex. 1004, ¶ 151.

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

This limitation includes multiple parts, each of which is disclosed by Mori. Ex. 1004, ¶ 152.

Generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points. Mori discloses this portion of element [1.b]. *Id.* ¶ 153–54. Mori explains that after the “word extraction unit . . . extracts the printed and readable byte strings as words from the Object recorded above, and uses it as a word set,” that “*a feature vector*” is constructed “for each Object based on the above word set.” Ex. 1006, [0029]–[0030]. Mori also discloses that “known supervised learning methods can be used” can be used wherein “[t]eacher data . . . is given in advance,” which can then be used as “a highly accurate method for estimating [a] class . . . for the newly observed” feature vector. *Id.*, [0056]–[0057]; *see also id.*, [0031] (“The supervised machine learning system outlined above can be re-learned by providing feedback of new data each time an arbitrary configurable cycle arrives.”).

Based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files. Mori

discloses this portion of element [1.b]. Ex. 1004, ¶¶ 155–57. As explained above, Mori discloses that “known supervised learning methods” can be used wherein “teacher data . . . is given in advance,” and “is a highly accurate method for estimating [a] class” for a newly observed feature vector. Ex. 1006, [0056]–[0057]; *see also id.*, [0031] (“The supervised machine learning system outlined above can be re-learned by providing feedback of new data each time an arbitrary configurable cycle arrives.”).

Mori explains that, in one example, 100 known malware files and 100 known “normal files” were used for machine learning. *Id.*, [0062]. Mori further notes that, “not only can we determine whether the subject is malware or a normal file, but also use machine learning to classify *other classes* of malware and determine the malware type.” *Id.*, [0063]. These “malware” and “normal file” classification types correspond to the “known malicious executable files” and “known benign executable files,” respectively. Ex. 1004, ¶ 156. Likewise, one of skill in the art at the time of filing of the ’844 Patent would have understood Mori’s disclosure of “other classes” of malware to include, for example, adware that is not necessarily “malicious,” but nevertheless comprises a class of “known unwanted executables” as recited by this limitation. *Id.*, ¶ 157 (explaining that a POSITA would have understood Mori’s disclosure of “other classes of malware” to comprise non-malicious but unwanted adware).

Wherein the collection of data comprises at least a portion of the plurality of static data points. Mori discloses this portion of element [1.b]. Ex. 1004, ¶ 158. In one example, the word extract unit “uses [a] word dictionary . . . prepared in advance,” and “extract[s] only those words that match the words in [the] word dictionary.” Ex. 1006, [0050]. And, the feature vectors that are utilized for supervised machine learning likewise use the same “word dictionary.” *Id.*, [0060]–[0062].

Wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range. Mori discloses this portion of element [1.b]. Ex. 1004, ¶¶ 159–65. Mori generates a feature vector for each Object based on a “word set” comprising the “printed and readable byte strings [extracted] as words from the Object” using, for example, “a static dictionary provided in advance.” Ex. 1006, [0029]–[0032]. Mori explains that the feature vector can be defined “as whether a word existed or not by assigning it ‘0’ or ‘1.’” *Id.*, [0030]. In other words, features are selectively turned on or off based on whether or not a word in the predetermined dictionary is present in the Object or not. *See* Ex. 1004, ¶ 160.

Mori provides an example wherein the “[w]ord extract unit . . . uses word dictionary . . . prepared in advance for the word candidates shown in Table 2 above,

extracting only those words that match the words in [the] word dictionary.” Ex. 1006, [0049]–[0050]. Mori further explains that, alternatively, “an Internet search engine is used to search for appropriate word candidates, with words extracted only when the number of search results exceeds an arbitrary configurable threshold.” Id., [0050].

Mori discloses the use of “known tf-idf values” as a configuration example for the generated feature vectors. A so-called “tf-idf value” (*i.e.*, term frequency-inverse document frequency) is a measure that reflects the importance or relevance of string representations in a single document amongst a collection (or corpus) of documents. Ex. 1004, ¶ 162. “Term frequency” refers to the number of times the word appears in a document, and “inverse document frequency” represents how common (or uncommon) a word is among the corpus. Multiplying these values together results in tf-idf values that have higher scores for more important/relevant terms, and lower scores (approaching zero) as a term becomes less relevant. *Id.*; see Ex. 1006, [0053] (explaining that “known tf-idf values can be used a configuration example of v_{ij} ,” or “ $v_{ij} = tf_{ij} * idf_{ij}$ ” for each word w_j in a file). Mori’s Table 4 depicts the “result of analyzing the words that appear in each file as well as the word counts for each file”:

Table 4

File name (hash value)	Appearance of word: number of times
00c4fc2fd0ac1d2b2fa43d8c5a087104	dll:4,get:2,sip:2,kernel:2,address:1,change:1,...
013a5ba10e3fc8a039b045530381d957	dll:4,get:2,kernel:2,rhd:1,bde:1,amu:1,range:1,...
notepad.exe	get:47>window:20,set:19,file:16,message:13,...
****	****

Id., [0059], [Table 4]. Mori’s Table 5 depicts “an example of configuring feature vectors” based on Table 4, wherein each word is numbered based on the word dictionary:

Table 5

File ID	Feature Vector	Class
1	119:0.0008, 196:0.0077, 307:0.0077,...	-1
2	119:0.0024, 168:0.0051, 213:0.0382,...	-1
3	84:0.006, 94:0.002, 113:0.00258,...	+1
****	****	****

Id., [0060], [Table 5]. As shown in Table 5, words that are present in each file are represented as “{word number: tf-idf value}” in the feature vector. Thus, if a word is not present, the “static data point” for that word is “turned off” since it is not included in the generated feature vector for that file (*i.e.*, the tf-idf value is 0 because the tf portion is 0). Ex. 1004, ¶ 165.

[1.c]: evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

Mori discloses [1.c]. Ex. 1004, ¶¶ 166–68. Mori describes the use of “support vector machines (SVMs)” as a “widely used” and “known” supervised learning method for estimating a class from an observed feature vector. Ex. 1006, [0057]–[0058]. Mori’s Table 6 illustrates an “example of the results of applying SVM to the above feature vector,” and shows 221 files classified as malware, and 1526 files classified as “normal” files. *Id.*, [0061]–[0063]. One of skill in the art at the time of filing of the ’844 Patent would have understood Mori’s disclosure of using and applying a “support vector machine (SVM)” to be synonymous with “using support vector processing” as required by element [1.c]. See Ex. 1004, ¶ 167.

b. Claim 8

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

Mori discloses this additional element. Ex. 1004, ¶¶ 169–70. Mori describes “determining whether [an] Object is malware based on newly observed feature vectors by applying *a machine learning system* to the feature vectors.” Ex. 1006, [0010]; see *id.*, [0055] (“By applying the supervised machine learning system . . . to the feature vector . . ., the file is classified as ‘malware,’ [or] ‘normal file.’”).

c. Claim 9

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

Mori discloses this additional element. Ex. 1004, ¶¶ 171–72. Mori explains that the “[m]achine learning system . . . can also perform re-learning via feedback of new data upon each arbitrary configurable cycle.” Ex. 1006, [0064]. Mori explains that such “new data” refers to “data about files that are not real-time, such as through prior art, or that, over time, have been determined to be malware or normal files.”

Id.

d. Independent Claim 10

[10.pre]: A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:

See element [1.pre], above.

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

See element [1.a], above.

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

See element [1.b], above.

[10.c]: evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

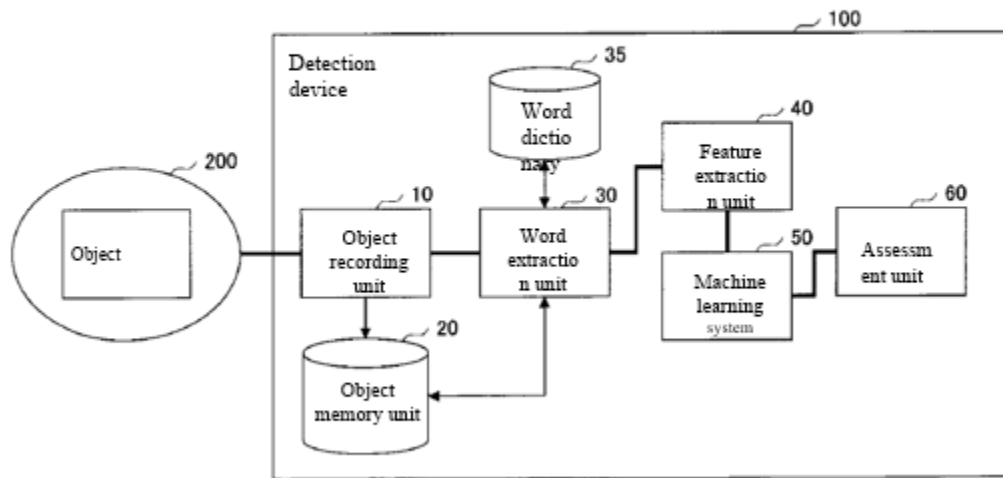
See element [1.c], above.

e. Independent Claim 17

[17.pre]: A system comprising: at least one memory; and at least one processor operatively connected with the memory and configured to perform operation of:

To the extent the preamble of claim 17 is limiting, Mori discloses [17.pre].

Ex. 1004, ¶¶ 178–79. Mori discloses “a system configuration,” as shown in Figure 1:



[Figure 1]
Example of a system configuration in the embodiment
of the present invention

Ex. 1006, [0036], Fig. 1. Mori explains that “[d]etection device 100 is connected to network 200 and consists of Object recording unit 10, Object storage unit 20, word extraction unit 30, word dictionary 35, feature extract unit 40, machine learning system 50, and assessment unit 60,” and that “the Object storage unit 20 and word dictionary 35 are storage media such as a hard disk.” *Id.*, [0037].

[17.a]: extracting a plurality of predefined character strings from an executable file without decrypting or unpacking the executable file;

See element [1.a], above.

[17.b]: generating a feature vector from the plurality of predefined character strings using a classifier trained to classify the plurality of predefined character strings based on a collection of data comprising known malicious executable files, known benign executable files and known unwanted executable files, wherein the collection of data comprises at least a portion of one or more of the plurality of predefined character strings, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more predefined character strings from the plurality of predefined character strings is within a predetermined range; and

See element [1.b], above.

[17.c]: evaluating the feature vector using support vector processing to determine whether the executable file is harmful.

See element [1.c], above.

C. Ground 3: Claims 1–20 Are Rendered Obvious Over Ideses in View of Mori

1. Overview of Ground

As demonstrated in Ground 1 Ideses discloses (or renders obvious) all of the limitations of the Challenged Claims. And as demonstrated in Ground 2, Mori renders obvious claims 1, 8-10, and 17. Ex. 1004, ¶ 185. To the extent that Patent Owner argues that Ideses does not sufficiently teach or suggest any element of any of the challenged independent claims, Mori discloses these features. Thus, the combination of Ideses and Mori renders obvious claims 1–20 of the '844 Patent.

2. Analysis of Unpatentability and Motivation to Combine

It would have been obvious to a POSITA at the effective filing date of the '844 Patent to have combined Mori's "readable strings" malware analysis technique with Ideses' malware detection method. Ex. 1004, ¶ 184. Specifically, a POSITA

reading Ideses would have been motivated to incorporate Mori's technique to enable Ideses' method to be applied to encrypted and/or packed executable files without decrypting or unpacking those files. *Id.*

Except for the specific limitations and claims discussed below, Petitioners incorporate by reference the analysis of the Ideses and Mori grounds above. See Sections IX.A.2 and IX.B.2, *supra*.

Extracting . . . without decrypting or unpacking the executable file (elements [1.a], [10.a], and [17.a]). As explained above, Mori discloses a malware detection technique that “focuses on readable strings contained in software files” and that is distinguishable from other techniques that require decrypting or unpacking the executable file, which “are not suitable for use in detecting communications containing malware in real time.” Ex. 1006, [0007], [0023]. In the “Problem to be Solved by the Invention” section of Mori, it is noted that prior art malware detection methods “that use decryption, disassembly, and debuggers are time-consuming and require computational resources.” *Id.*, [0007]. As an example, Mori identifies “an approach to detecting whether malware is present by executing it on an actual operating system and analyzing its behavior” as “time-consuming and requiring computational resources, meaning that timely detection and determination could not be performed.” *Id.*, [0008]; Ex. 1004, ¶¶ 186–88.

Ideses likewise recognizes that certain “resource limited” mobile devices may be “unable to extract the full set of [classification] features during runtime, and/or unable to perform the extraction within a reasonable amount of time to allow for run-time operation.” Ex. 1005, [0060]. Thus, a POSITA would have been motivated to combine the Ideses method with Mori’s “readable strings” technique, which is an offline (*i.e.*, static analysis) technique, in order to enable a resource limited client device to perform the feature extraction and malware analysis locally. Ex. 1004, ¶¶ 189–90.

A POSITA would have had a reasonable expectation of success because both Ideses and Mori describe offline/static malware analysis techniques. *See, e.g.*, Ex. 1005, [0090] (“The feature extraction may take place off-line, not part of a run-time operation.”); Ex. 1006, [0008]–[0009] (distinguishing “an approach to detecting whether malware is present by executing it on an actual operating system and analyzing its behavior” from the “present invention”). Moreover, the combination of Ideses and Mori would have been nothing more than the use of a known technique (Mori’s focus on “readable strings” for offline malware detection) to improve similar devices (Ideses’ malware detection system) in the same way to yield predictable results (the ability to detect malware without decrypting or unpacking an executable). Ex. 1004, ¶ 191.

Predefined character strings (elements [1.a], [10.a], [17.a], and [17.b]). As explained above, Mori “focuses on *readable strings contained in software files*, collects information (characteristic vectors) that characterizes malware, such as messages contained in programs that constitute malware, host names, execution command names, function names and library names used for communication,” and “compares this content with strings contained in files being inspected to detect malware.” *Id.*, [0023]. Accordingly, Mori explicitly discloses the extraction of “predefined character strings” as required by these claim elements. See Section IX.B.2.a, *supra*; Ex. 1004, ¶ 192–93.

Ideses likewise explains that “the extracted features may contain *static meta data* regarding the application, for example, icon, name, rating or other features,” *Id.*, [0089]. A POSITA would have been motivated to use Mori’s examples of predefined “readable strings . . . such as messages, . . . host names, execution command names, function names and library names” as additional features for extraction in the Ideses method, for the same reasons described above. Ex. 1004, ¶¶ 194–95.

X. PTAB DISCRETION SHOULD NOT PRECLUDE INSTITUTION

A. PTAB Should Not Exercise Its Discretion to Deny Institution Under *Fintiv*

Consistent with congressional intent and the goals of *NHK/Fintiv*, Petitioners took all possible steps to promote the Board’s efficiency and fairness objectives.

Petitioners initiated this petition with diligence, filed a single, focused petition, and served a stipulation (Ex. 1003) broader than one the Board previously found mitigated concerns of duplicative efforts between the district court and the Board and potentially conflicting decisions. *Sand Revolution II, LLC v. Continental Intermodal Grp.*, IPR2019-01393, Paper 24 (June 16, 2020). These facts, along with the merits of Grounds 1–3, strongly favor institution.

1. Factor 1: Likelihood of Stay

Institution will enable the Board to resolve the issue of validity, and a finding of invalidity will relieve the District Court of the need to thereafter continue with the litigation. The District Court has made clear in prior cases that a motion to stay prior to institution of an IPR would be premature, and the Board recently “decline[d] to speculate on the likelihood of Judge Albright entering a stay after institution” and thus found Factor 1 neutral. *Western Digital Corp. v. Kuster*, IPR2020-01391, Paper 10 at 9 (Feb. 16, 2021); *see also Volkswagen Grp. of Am., Inc. v. Arigna Tech. Ltd.*, IPR2021-01321, Paper 10 at 7 (Feb. 15, 2022) (declining to speculate about the likelihood of a stay). Thus, this factor is at worst neutral.

2. Factor 2: District Court Schedule

The District Court schedule weighs in favor of institution. Pursuant to the District Court’s amended scheduling order on January 22, 2022 (Dkt. No. 160), the trial in this case will not begin until August 19, 2024. Ex. 1008, 6. Based on an 18-

month IPR schedule, even if the trial began exactly as currently projected, a Final Written Decision (“FWD”) arising from this petition would still be likely to issue before trial.

Even if that were not the case, uncertainty about the trial date and the high likelihood of further delay weigh against discretionary denial. In general, District Court patent-validity litigation schedules are uncertain. Indeed, the Board has held that this factor was “neutral” even in cases where the projected trial date was before the Board’s statutory final-decision deadline because it has recognized that projected trial dates are uncertain, especially when projected in the early stages of litigation. In *Cizion, LLC v. Kerr Machine Co.*, for instance, trial was set “three months prior to [the Board’s] (anticipated) deadline for a final written decision.” PGR2020-00065, Paper 10 at 26 (Dec. 3, 2020). The Board explained that “[a]lthough the current schedule [in *Cizion*] indicates that trial will commence a few months prior to the statutory due date to issue a final written decision in this proceeding, evidence in the record cited by the Petitioner does indicate that the actual trial date is uncertain.” *Id.*; see also *Micron Tech., Inc. v. Godo Kaisha IP Bridge 1*, IPR2020-01008, Paper 10 at 14 (Dec. 7, 2020) (“Accordingly, even if we consider the Texas court to have set a formal trial date for a few months before the final written decision, due to the uncertainty as to this trial date, this factor is, at most, neutral to whether we should exercise our discretion to deny the Petition.”).

Of particular relevance here, the *Cizion* Board credited the Petitioner’s evidence that “in the Western District of Texas, 70% of trial dates have slipped following PTAB discretionary denials.” PGR2020-00065, Paper 10 at 26 (Dec. 3, 2020). The Board further noted that the evidence included “multiple proceedings with trial dates close to the” same trial date. *Id.* The same is true here, where the District Court has set a single trial date for five separate litigations with dozens of asserted patents. Ex. 1008.

Thus, the general uncertainty of District Court patent-validity actions, and the high likelihood that this litigation will extend beyond its projections, weighs in favor of institution. Indeed, recent Interim Guidance⁵ dictates that because scheduled trial dates “are unreliable and often change,” a scheduled trial date “is not by itself a good indicator of whether the district court trial will occur before the statutory deadline for a [FWD].” *Samsung Elec.’s Co., Ltd. et al v. Staton Techniya, LLC*, IPR2022-00302, Paper 13, 10 (PTAB July 11, 2022).

Finally, trial is more likely to get pushed further back in this case because the litigation is especially complex. Patent Owner originally asserted nine patents against five defendants, including the ’844 Patent, and recently moved to assert five

⁵ https://www.uspto.gov/sites/default/files/documents/interim_proc_discretionary_denials_aia_parallel_district_court_litigation_memo_20220621_.pdf

more patents against Sophos Ltd. and other defendants. In granting Patent Owner’s motion to amend, the Court acknowledged that this case is “about as complicated as you’re going to get with patent cases” and said it would be “amenable to adjusting the schedule” by “granting more time to address things.” Ex. 1018, 2. Because this multi-defendant consolidated case is exceptionally complicated, it is unlikely that the District Court will dispose of the dispute over the ’844 Patent before a FWD would issue.

Thus this factor weighs strongly against discretionary denial.

3. Factor 3: Petitioners’ Investment in IPR Outweighs Forced Investment in Litigation to Date

Petitioners’ substantial investment in these IPR proceedings should counterbalance—if not outweigh—the resources invested in the co-pending litigation. It would be unjust to consider resources expended in the District Court (equally by both parties), without considering Petitioners’ resources expended to prepare this IPR petition that would be irretrievably lost without consideration on the merits, in addition to the extensive expenses that may be foregone through institution of this petition and that will otherwise certainly follow in co-pending litigation for which significant milestones exist. *See, e.g., Apple v. Seven Networks*, IPR2020-00255, Paper 13 at 12–15 (July 28, 2020) (declining to exercise discretionary denial on petitions filed nine months after the commencement of co-

pending litigation where a large number of patents were included in the litigation, and a large number of claims from each patent were challenged in the petitions).

In *Sand Revolution*, even after invalidity contentions and the Markman hearing, the Board found that this factor weighed “only marginally, if at all, in favor of exercising discretion.” *Sand Revolution*, IPR2019-01393, Paper 24, at 11. Given Petitioners’ diligence and significant investment in filing this petition, especially this early in the litigation, this factor should weigh against a discretionary denial.

4. Factor 4: The Petition Raises Unique Issues

Factor 4 strongly supports institution. Consistent with *Fintiv*, Petitioners ask the Board to consider the unique challenges raised in the petition. Petitioners have stipulated that, if the Board institutes the present petition, Petitioners will not assert that the challenged claims are invalid on the instituted grounds in the related litigation, nor any other possible prior art printed publication grounds based on the references in Grounds 1–3. *Webroot, Inc. et al. v. Sophos Ltd.*, No. 6:22-cv-00240-ADA-DTG (W.D. Tex.). This stipulation is significantly broader than the stipulation in *Sand Revolution* and therefore this factor strongly supports institution. *Sand Revolution*, Paper 24 at 11.

5. Factor 5: Institution Will Provide the Board an Opportunity to Invalidate Claims That Later Could Be Reasserted Against Others

In the District Court litigation, Patent Owner has asserted 15 patents against five defendants. Of those 15 patents, three are asserted against all five defendants (including the '844 Patent); four are asserted against four defendants; and the average patent is asserted against 2.9 defendants. *See Webroot, Inc. et al. v. Forcepoint LLC, Case No. 6:22-CV-00243 (W.D. Tex.)*. Thus, Patent Owner has a demonstrated history of asserting claims against multiple defendants, and the Board has an opportunity to invalidate some of those claims here. Factor 5 weighs in favor of institution.

6. Factor 6: Other Circumstances Support Institution

As explained above, the merits of the unpatentability grounds are particularly strong, weighing in favor of institution. *See Fintiv*, 14–15. And the effect on “the economy [and] the integrity of the patent system” further supports institution. PTAB Consolidated Trial Practice Guide, 56. The claims of the '844 Patent are very broad and purport to cover extracting character strings from an executable file and using machine learning to detect malware, which was already disclosed several years prior to even the effective filing date of the patent. The Board should consider and invalidate the claims such that they may not be asserted against other entities in the future.

**B. PTAB Should Not Exercise its Discretion To Deny Institution
Under *Becton* and *Advanced Bionics***

Institution should not be denied under the framework set forth in *Becton, Dickinson & Co. v. B. Braun Melsungen AG*, IPR2017-01586, Paper 8 (Dec. 15, 2017) (precedential as to § III.C.5, first paragraph) and *Advanced Bionics LLC v. MED-EL Elektromedizinische Gerate GmbH*, IPR2019-01469, Paper 6 (PTAB Feb. 13, 2020) (precedential). Neither Ideses nor Mori was before the Examiner during prosecution. Therefore, the arguments presented in this Petition are not the same as, or substantially similar to, arguments previously presented to the Office. Thus, the first prong of the *Advanced Bionics* framework is not met. And because the first prong is not met, there is no need to reach the second prong to resolve against discretionary denial under Section 325(d)—though in any event, there are no pending proceedings before the PTAB concerning the '844 Patent, so the second prong is not met either. *Oticon Medical AB v. Cochlear Ltd.*, IPR2019-00975 Paper 15, 20 (PTAB Oct. 16, 2019) (precedential).

XI. CONCLUSION

Petitioners have established a reasonable likelihood that the Challenged Claims are unpatentable. Petitioners therefore respectfully request that IPR be instituted.

DATED: February 7, 2023

Respectfully Submitted,

By: /s/ Ryan Iwahashi

Ryan Iwahashi
(Reg. No. 63,378)
Gibson, Dunn & Crutcher LLP
1881 Page Mill Road
Palo Alto, CA 94304-1211
Tel: (650) 849-5367
RIwahashi@gibsondunn.com

*Attorney for Petitioners Sophos Ltd. and
Sophos Inc.*

CERTIFICATION UNDER 37 C.F.R. § 42.24(d)

Under the provisions of 37 C.F.R. § 42.24(d), the undersigned attorney hereby certifies that the word count for Sections I and III–XI of the foregoing Petition for *Inter Partes* Review is 13,957, according to the word count tool in Microsoft Word.

DATED: February 7, 2023

By: /s/ Ryan Iwahashi

Ryan Iwahashi
(Reg. No. 63,378)
Gibson, Dunn & Crutcher LLP
1881 Page Mill Road
Palo Alto, CA 94304-1211
Tel: (650) 849-5367
RIwahashi@gibsondunn.com

*Attorney for Petitioners Sophos Ltd. and
Sophos Inc.*

CERTIFICATE OF SERVICE

The undersigned certifies service pursuant to 37 C.F.R. §§ 42.6(e) and 42.105(a), (b) on the Patent Owner via FedEx overnight mail of a copy of this Petition for *Inter Partes* Review and supporting materials at the correspondence address of record for the '844 Patent shown in USPTO Patent Center:

Sprinkle IP Law Group
1301 W. 25th Street, Suite 400
Austin, TX 78705

Additionally, a courtesy copy has been mailed to Patent Owner at:

Webroot Inc.
385 Interlocken Crescent, Suite 800
Broomfield, CO 80021

DATED: February 7, 2023

By: /s/ Ryan Iwahashi

Ryan Iwahashi
(Reg. No. 63,378)
Gibson, Dunn & Crutcher LLP
1881 Page Mill Road
Palo Alto, CA 94304-1211
Tel: (650) 849-5367
RIwahashi@gibsondunn.com

*Attorney for Petitioners Sophos Ltd. and
Sophos Inc.*

APPENDIX: CHALLENGED CLAIM LISTING

No.	Limitation
[1.pre]	A computer-implemented method comprising:
[1.a]	extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file;
[1.b]	generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files, and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more static data points from the plurality of extracted static data points is within a predetermined range; and
[1.c]	evaluating the feature vector using support vector processing to determine whether the executable file is harmful.
[2]	The computer-implemented method according to claim 1, wherein the extracting the plurality of static data points further comprises classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classification.
[3]	The computer-implemented method according to claim 2, wherein generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.
[4]	The computer-implemented method according to claim 2, wherein the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.

No.	Limitation
[5]	The computer-implemented method according to claim 4, wherein analyzing the executable file further comprises generating at least one feature vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.
[6]	The computer-implemented method according to claim 1, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.
[7]	The computer-implemented method according to claim 6, wherein the threshold value is set based on predetermined false positive range data.
[8]	The computer-implemented method according to claim 1, wherein the plurality of static data points is extracted using a machine learning technique.
[9]	The computer-implemented method according to claim 1, wherein a determination of whether the executable file is harmful is used to retrain the classifier.
[10.pre]	A computer-readable storage device containing instructions, that when executed on at least one processor, causing the processor to execute a process comprising:
[10.a]	extracting a plurality of static data points from an executable file without decrypting or unpacking the executable file, wherein the plurality of static data points represent predefined character strings in the executable file;

No.	Limitation
[10.b]	generating a feature vector from the plurality of static data points using a classifier trained to classify the plurality of static data points based on a collection of data comprising known malicious executable files, known benign executable files and known unwanted executable files, wherein the collection of data comprises at least a portion of the plurality of static data points, and wherein one or more features of the feature vector are selectively turned on or off based on whether one or more values of one or more static data points from the plurality of extracted static data points is within a predetermined range; and
[10.c]	evaluating the feature vector using support vector processing to determine whether the executable file is harmful.
[11]	The computer-readable storage device according to claim 10, wherein the extracting of the plurality of static data points executed by the processor further comprises classifying the extracted static data points according to a type of data extracted from the executable file, and encoding the extracted static data points for the classifier based on the classifying.
[12]	The computer-readable storage device according to claim 11, wherein generating the at least one feature vector comprises selectively setting features of the classifier based on the plurality of extracted static data points.
[13]	The computer-readable storage device according to claim 11, wherein the classifying the at least one static data point comprises classifying the plurality of extracted static data points into categories comprising numeric values, nominal values, string or byte sequences, and Boolean values.
[14]	The computer-readable storage device according to claim 13, wherein analyzing the executable file further comprises generating a feature vector from the plurality of extracted static data points, wherein features of the generated feature vector are weighted based at least upon the classified categories and the plurality of extracted static data points.

No.	Limitation
[15]	The computer-readable storage device according to claim 10, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.
[16]	The computer-readable storage device according to claim 15, wherein the threshold value is set based on predetermined false positive range data.
[17.pre]	A system comprising: at least one memory; and at least one processor operatively connected with the memory and configured to perform operation of:
[17.a]	extracting a plurality of predefined character strings from an executable file without decrypting or unpacking the executable file;
[17.b]	generating a feature vector from the plurality of predefined character strings using a classifier trained to classify the plurality of predefined character strings based on a collection of data comprising known malicious executable files, known benign executable files and known unwanted executable files, wherein the collection of data comprises at least a portion of one or more of the plurality of predefined character strings, and wherein one or more features of the feature vector are selectively turned on or off based on whether a value of one or more predefined character strings from the plurality of predefined character strings is within a predetermined range; and
[17.c]	evaluating the feature vector using support vector processing to determine whether the executable file is harmful.
[18]	The system according to claim 17, wherein the determining further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value, and wherein the threshold value is set based on predetermined false positive range data.

No.	Limitation
[19]	The system according to claim 17, wherein the extracting the plurality of predefined character strings further comprises classifying the extracted plurality of predefined character strings according to a type of data extracted from the executable file, and encoding the extracted plurality of predefined character strings for the classifier based on the classification.
[20]	The system according to claim 17, wherein determining whether the executable file is harmful further comprises preventing execution of the executable file when a determined probability value that the executable file is harmful exceeds a threshold value.