

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

PALO ALTO NETWORKS, INC.,
Petitioner,

v.

CENTRIPETAL NETWORKS, INC.,
Patent Owner.

Case IPR2021-01155
Patent No. 10,567,343

PETITION FOR *INTER PARTES* REVIEW

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	MANDATORY NOTICES UNDER 37 C.F.R. § 42.8.....	2
	A. Real Party-In-Interest	2
	B. Related Matters.....	3
	C. Lead and Back-up Counsel, and Service Information	3
III.	PAYMENT OF FEES	3
IV.	REQUIREMENTS FOR <i>INTER PARTES</i> REVIEW.....	4
	A. Grounds for Standing	4
	B. Identification of Challenge.....	4
	1. The Specific Art on Which the Challenge is Based	4
	2. Statutory Grounds on Which the Challenge is Based.....	6
	C. 314(a) Discretion Does Not Apply.....	7
	D. 325(d) Discretion Does Not Apply	8
V.	BACKGROUND	12
	A. Summary of the '343 Patent.....	12
	B. Prosecution History of the '343 Patent	21
VI.	LEVEL OF ORDINARY SKILL IN THE ART.....	23
VII.	CLAIM CONSTRUCTION	24
	A. “operator”	24
VIII.	GROUND OF UNPATENTABILITY.....	25
	A. Ground 1: Claims 1-6, 8-13, and 15-19 Are Rendered Obvious Over Sourcefire.....	26
	1. Overview of Sourcefire.....	26

2.	Claim Chart	34
B.	Ground 2: Claims 7, 14, and 20 Are Rendered Obvious Over Sourcefire in View of Mantripragada	70
1.	Overview of Mantripragada	70
2.	Motivation to Combine	71
IX.	SECONDARY CONSIDERATIONS	73
X.	CONCLUSION.....	73

LIST OF EXHIBITS

Exhibit ("Ex.")	Description
1001	U.S. Patent No. 10,567,343 (the "'343 Patent")
1002	File History for U.S. Patent No. 10,567,343
1003	Declaration of Dr. Robert Akl, D.Sc. ("Akl")
1004	Sourcefire 3D System User Guide Version 4.10 ("Sourcefire")
1005	Declaration of John Leone
1006	U.S. Pat. No. 7,032,031 ("Jungck")
1007	U.S. Pat. App. Publication 2009/0103524 ("Mantripragada")
1008	U.S. Pat. No. 8,370,936 ("Zuk")
1009	U.S. Pat. No. 7,900,042 ("Hall")
1010	SSL Inspection – Sourcefire Cybersecurity, archived at Internet Archive Wayback Machine on December 20, 2010 at https://web.archive.org/web/20101220151851/http://www.sourcefire.com/security-technologies/cyber-security-products/3d-system/sslencryption-decryption ("SSL Appliances Webpage").
1011	Sourcefire SSL Appliances data sheet, archived at Internet Archive Wayback Machine on March 26, 2012 at https://web.archive.org/web/20120316165725/http://www.ndm.net:80/sourcefirestore/downloads/Sourcefire%20SSL%20Appliance%20Fact%20Sheet%20%28English%29.pdf ("SSL Appliances Data Sheet").
1012	U.S. Patent No. 7,586,899 ("Mohaban")
1013	Bard, G.V., "Cryptology ePrint Archive: Report 2006/136: A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL," archived at Internet Archive Wayback Machine on May 3, 2006 at https://web.archive.org/web/20060503043443/http://eprint.iacr.org:80/2006/136 .
1014	Bard, G.V., "A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL," archived at Internet Archive Wayback Machine on May 3, 2006 at https://web.archive.org/web/20060503043443/http://eprint.iacr.org:80/2006/136
1015	T. Dierks & E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1," <i>IETF RFC 4346</i> , April 2006, archived at Internet Archive Wayback Machine on June 26, 2006 at https://web.archive.org/web/20060626134135if_/http://www.ietf.org:8

	0/rfc/rfc4346.txt.
1016	S. Turner & C. Polk, “Prohibiting Secure Sockets Layer (SSL) Version 2.0,” <i>IETF RFC 6176</i> , March 2011, archived at Internet Archive Wayback Machine on March 18, 2011 at https://web.archive.org/web/20110318184421/https://tools.ietf.org/html/rfc6176 .
1017	NDM Company (“Sourcefire Authorized Distributor”) Product Data Sheets Webpage, archived at Internet Archive Wayback Machine on October 28, 2011 at https://web.archive.org/web/20111028104123/http://www.ndm.net:80/sourcefirestore/Product-Data-Sheets/product-data-sheets (“NDM Webpage”).
1018	A. Shostack, An Overview of SSL (version 2), May 1995, available from https://adam.shostack.org/ssl.html (accessed July 11, 2018), archived at Internet Archive Wayback Machine on May 25, 1997 at https://web.archive.org/web/19970525224610/http://www.homeport.org/~adam/ssl.html .
1019	U.S. Patent No. 9,055,093 (“Borders”)
1020	Barry M. Leiner <i>et al.</i> , “A Brief History of the Internet”. <i>Internet Society</i> , 1997, available at: https://www.internetsociety.org/internet/history-internet/brief-historyinternet/ .
1021	“Internet Protocol,” <i>IETF RFC 791</i> , J. Postel, ed., September 1981, archived at Internet Archive Wayback Machine on February 4, 2007 at https://web.archive.org/web/20070204151303/https://tools.ietf.org/html/rfc791 .
1022	“Transmission Control Protocol,” <i>IETF RFC 793</i> . J. Postel, ed., September 1981, archived at Internet Archive Wayback Machine on February 2, 2007 at https://web.archive.org/web/20070202201546/https://tools.ietf.org/html/rfc793 .
1023	E. H. Spafford, “The Internet Worm Program: An Analysis,” <i>Purdue Technical Report CSD-TR-823</i> , 1988.
1024	Eichin, M.W. and Rochlis, J.A., “With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988,” <i>Proceedings, 1989 IEEE Symposium on Security and Privacy</i> , (pp. 326-343), IEEE, 1989.
1025	T. Eisenberg <i>et al.</i> , “The Cornell Commission: On Morris and the Worm,” <i>Communications of the ACM</i> , Volume 32 Number 6, June 1989.

1026	J. C. Mogul, “Simple and Flexible Datagram Access Controls for Unix-based Gateways,” <i>Digital Western Research Laboratory Technical Report No. 89/4</i> , 1989.
1027	Heberlein, L. T.L., <i>et al.</i> , “A Network Security Monitor,” <i>Proceedings, 1990 IEEE Computer Society Symposium on Security and Privacy</i> , 1990.
1028	“NetRanger Network Security Management System User Guide”. v1.3.1 <i>Wheelgroup Corporation</i> , 1997.
1029	Roesch, M., “Snort: Lightweight Intrusion Detection for Networks,” <i>Lisa</i> (Vol. 99, No. 1, pp. 229-238), 1999.
1030	Brandon De Hoyos, “IRC, ICQ, AIM and More: A History of Instant Messaging,” <i>Lifewire</i> , Oct 2016, available from https://www.lifewire.com/im-a-brief-history-1949611 .
1031	M. Chatel, “Classical vs. Transparent IP Proxies,” <i>IETF RFC 1919</i> , March 1996.
1032	K. Ingham & S. Forrest, “A History and Survey of Network Firewalls,” <i>University of New Mexico Computer Science Department Technical Report 2002-37</i> , 2002.
1033	Sourcefire Support Website Webpage, archived at Internet Archive Wayback Machine on April 30, 2011 at https://web.archive.org/web/20110430194135/https://support.sourcefire.com/ (“Sourcefire Support Webpage”).
1034	Business Wire, “Sourcefire Expands IPS Solutions Portfolio, Adding FirePOWER to the Industry’s Leading Protection,” April 18, 2011, available at https://www.businesswire.com/news/home/20110418005189/en/Sourcefire-Expands-IPS-Solutions-Portfolio-Adding-FirePOWER (accessed June 20, 2018)
1035	“File Transfer Protocol,” IETF RFC 765, J. Postel, ed., June 1980, archived at Internet Archive Wayback Machine on February 6, 1997 at https://web.archive.org/web/20070206005843/https://tools.ietf.org/html/rfc765 .
1036	T. B. Lee, “The Original HTTP as defined in 1991,” World Wide Web Consortium, 1991, archived at Internet Archive Wayback Machine on June 5, 1997 at https://web.archive.org/web/19970605071155/https://www.w3.org/Protocols/HTTP/AsImplemented.html .
1037	I. Ristic, “SSL/TLS and PKI History,” April 2018, available from https://www.feistyduck.com/ssl-tls-and-pki-history/ .

1038	E. Rescorla, “HTTP over TLS”, <i>IETF RFC 2818</i> , May 2000, archived at Internet Archive Wayback Machine on January 25, 2007 at https://web.archive.org/web/20070125003236/https://tools.ietf.org/html/rfc2818 .
1039	J. Moser, “The First Few Milliseconds of an HTTPS Connection”, <i>Moserware</i> , June 10, 2009, archived at Internet Archive Wayback Machine on June 13, 2009 at https://web.archive.org/web/20090613041022/http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html .
1040	U.S. Patent No. 9,124,552 (the “’552 Patent”)
1041	IPR2018-01436, Pap. 7 (Jan. 24, 2019) (Institution Decision for U.S. Patent No. 9,124,552)
1042	IPR2018-01436, Pap. 40 (Jan. 23, 2020) (Final Written Decision for U.S. Patent No. 9,124,552)
1043	<i>Centripetal Networks, Inc. v. Cisco Sys., Inc.</i> , 847 F. App’x 869 (Fed. Cir. 2021)
1044	Chart Comparing the Challenged Claims to the Claims of U.S. Patent No. 9,124,552
1045	IPR2018-01559, Pap. 7 (April 2, 2019) (Institution Decision for U.S. Patent No. 9,686,193)
1046	June 28, 2021 Email from the U.S. District Court for the Eastern District of Virginia to Counsel
1047	July 1, 2021 Letter to the U.S. District Court for the Eastern District of Virginia
1048	June 30, 2021 Email from Counsel for Centripetal Networks, Inc.
1049	Declaration of Jonathan Bradford

I. INTRODUCTION

Petitioner Palo Alto Networks, Inc. (“Petitioner”) respectfully requests *inter partes* review of Claims 1-20 (“Challenged Claims”) of U.S. Patent No.

10,567,343 (the “’343 patent”) in accordance with §§311-319 and §42.100 et seq.¹

The Challenged Claims are unpatentable for the same reasons that, in a prior IPR, the Board found similar claims of U.S. Patent 9,124,552 (“’552 patent”), the grandparent to the ’343 patent, to be unpatentable over the prior art relied upon herein. Ex. 1042; *aff’d Centripetal Networks, Inc. v. Cisco Sys., Inc.*, 847 F. App’x 869 (Fed. Cir. 2021) (Ex. 1043).

The ’343 patent, like the invalid ’552 patent, discloses a two-stage packet filtering process for packets being transmitted between networks. In a first stage, the “5-tuple” (*e.g.*, IP addresses, TCP ports) packet-header-field values are filtered, and in a second stage, application packet-header-field values are filtered. The ’343 patent explains that application packet-header-field values can include encryption protocol version (*e.g.* TLS 1.0) and HTTP methods (*e.g.*, “PUT”, “GET”). The ’343 patent also explains that such two-stage packet filtering was useful to prevent exfiltration of sensitive data from a network.

¹ Section cites are to 35 U.S.C. or 37 C.F.R. as context indicates, and all emphasis/annotations added unless noted.

Before the earliest priority date of the '343 patent, however, the same two-stage packet filtering was already commonly known. For example, the prior art **Sourcefire** reference describes that same functionality. **Sourcefire** describes a two-stage packet filtering process that included first-stage filtering based on the 5-tuple, as well as second-stage filtering based on “keywords” that can include application-layer data such as encryption protocol version (*e.g.*, TLS 1.0) and HTTP methods (*e.g.*, GET, PUT). The **Sourcefire** reference relied upon herein is the same ground that led to findings of unpatentability as to all claims of the related '552 patent.

Thus, the purported “invention” of the '343 patent would have been obvious by the earliest possible priority date. Indeed, the Challenged Claims of the '343 patent are *broader* than the '552 claims already found unpatentable, because the '343 claims are not limited to HTTPS packets.

Petitioner therefore requests that the Board institute trial and find the Challenged Claims unpatentable pursuant to the Grounds proposed herein.

II. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8

A. Real Party-In-Interest

Pursuant to 37 C.F.R. § 42.8(b)(1), Petitioner identifies Palo Alto Networks, Inc. as real party-in-interest.

B. Related Matters

The '343 patent is currently the subject of district court litigation:

Centripetal Networks, Inc. v. Palo Alto Networks, Inc., Case No. (E.D. Virginia, filed March 12, 2021) (the “E.D.Va. suit”).

C. Lead and Back-up Counsel, and Service Information

Lead Counsel	Backup Counsel
Scott A. McKeown Reg. No. 42,866 ROPES & GRAY LLP 2099 Pennsylvania Avenue, NW Washington, D.C. 20006-6807 Phone: 202-508-4740 Fax: 617-235-9492 scott.mckeown@ropesgray.com Mailing address for all PTAB correspondence: ROPES & GRAY LLP IPRM—Floor 43 Prudential Tower 800 Boylston Street Boston, Massachusetts 02199-3600	Mark Rowland Reg. No. 32,077 James Batchelder (<i>pro hac vice</i> forthcoming) Andrew Radsch (<i>pro hac vice</i> forthcoming) Christopher M. Bonny Reg. No. 63,307 ROPES & GRAY LLP 1900 University Ave., 6 th Floor East Palo Alto, CA 94303-2284 Phone: 650-617-4000 Fax: 617-235-9492 mark.rowland@ropesgray.com james.batchelder@ropesgray.com andrew.radsch@ropesgray.com christopher.bonny@ropesgray.com

Petitioner consents to electronic service of documents to the email addresses of the counsel identified above.

III. PAYMENT OF FEES

The undersigned authorizes the Office to charge the fee required by 37 C.F.R. §42.15(a) and any additional fees to Deposit Account No. 18-1945.

IV. REQUIREMENTS FOR *INTER PARTES* REVIEW

A. Grounds for Standing

Pursuant to 37 C.F.R. §42.104(a), Petitioner certifies that the '343 patent is available for *inter partes* review and that Petitioner is not barred or estopped from requesting *inter partes* review challenging the claims of the '343 patent on the grounds identified herein.

B. Identification of Challenge

Pursuant to 37 C.F.R. §§42.104(b) and (b)(1), Petitioner requests *inter partes* review of the Challenged Claims and that the Board cancel the same as unpatentable. The '343 patent is a continuation of a chain of applications with an earliest filing date of March 12, 2013, which Petitioner assumes is the priority date of the '343 patent.²

1. The Specific Art on Which the Challenge is Based

Petitioner relies upon the following prior art:

Exhibit 1004 – Sourcefire 3D System User Guide Version 4.10

(“**Sourcefire**”).

² Petitioner takes no position as to the propriety of the priority claim since the art presented herein pre-dates the earliest filing. Petitioner reserves the right to challenge this priority claim.

Sourcefire was publicly accessible at least as early as April 2011 and qualifies as prior art under §102(b),³ as it was enclosed on documentation disks (CD-ROM/DVD) included with each Sourcefire 3D System appliance subsequently sold (*e.g.*, 3D Sensor, Defense Center) by Sourcefire, Inc., and was publicly accessible for download by persons who had received a login and password from Sourcefire, Inc. to its support website. Ex. 1005; *see also* Exs. 1033-34. Based on this same evidence, the Federal Circuit expressly held that “Sourcefire was publicly accessible and therefore qualifies as a printed publication” under §102(b) relative to the same March 23, 2013 priority date as the ’343 patent. Ex. 1043, 5, 7-8; Ex. 1042, 12, 31. As a result, Patent Owner is barred by the doctrine of collateral estoppel from contesting, in this proceeding, whether **Sourcefire** qualifies as a printed publication.⁴

³ All references to §§102/103/112 are to pre-AIA 35 U.S.C., but the art relied upon herein is also prior art under the post-AIA framework.

⁴ “Collateral estoppel protects a party from having to litigate issues that have been fully and fairly tried in a previous action and adversely resolved against a party-opponent.” *Ohio Willow Wood Co. v. Alps South, LLC*, 735 F.3d 1333, 1342 (Fed. Cir. 2013); *MaxLinear, Inc. v. CF CRESPE LLC*, 880 F.3d 1373, 1376 (Fed. Cir. 2018) (estoppel applies in administrative context). Collateral estoppel is not limited

Ex. 1007 – U.S. Pat. App. Publication 2009/0103524 (“**Mantripragada**”),
published April 23, 2009, qualifies as prior art under §102(b).

2. Statutory Grounds on Which the Challenge is Based

Petitioner respectfully requests cancelation of the Challenged Claims on the
following grounds:

Ground	Statute	Claim(s)	Prior Art
1	§103	1-6, 8-13, 15-19	Sourcefire
2	§103	7, 14, 20	Sourcefire in view of Mantripragada

Pursuant to §42.204(b)(4), an explanation of the information required by
those sections is provided in Section VIII.

This Petition is supported by the Declaration of Dr. Robert Akl (Ex. 1003,
¶¶1-148) (“Akl”), which discusses background on the state of the art (*see* Akl

to identical patent claims; “[r]ather, it is the identity of the issues that were litigated
that determine whether collateral estoppel should apply.” *Ohio Willow Wood*, 735
F.3d at 1342. The Board regularly applies these principles in IPRs where similar
claims in related patents were adjudged to be unpatentable. *E.g.*, *Amazon v. M2M*,
IPR2019-01204, Pap. 43, *passim* (applying estoppel to issues adjudged in prior
IPRs involving related patent).

¶¶39-74), the scope and content of the prior art, and motivation to combine, at the time of the purported '343 patent invention.

C. 314(a) Discretion Does Not Apply

Co-pending district court proceedings do not warrant the exercise of discretion under §314(a) based on the six factors identified in *Apple Inc. v. Fintiv, Inc.*, IPR2020-00019, Paper 11. **1:** Petitioner intends to seek a stay of the E.D.Va. suit pending the institution of this IPR. E.D.Va. courts frequently grant stays pending IPR, including pre-institution. *E.g., RAI Strategic Holdings, Inc. v. Altria Clients Svcs.*, 1:20-cv-393, Dkt. 426 (E.D.Va. Dec. 4, 2020); *Centripetal Networks v. Cisco Sys.*, 2:18-cv-00094, Dkt. 58 (E.D.Va. Feb. 25, 2019); *Sharpe Innovations, Inc. v. T-Mobile USA*, 2:17-cv-351, Dkt. 41 (E.D.Va. Jan. 10, 2018). **2:** As of the filing of this Petition, the district court has not held a scheduling conference or set a trial date. Patent Owner continues to seek to add patents and claims to the case. Ex. 1048, 2. The court indicated that it may set trial for July 2022 (*see* Ex. 1046) and the parties indicated due to conflict that they would be available for an August 1, 2022 start date (*see* Ex. 1047), meaning trial would be at least one year away. **3:** The court has not issued any substantive orders related to the '343 Patent. Discovery has not started and no party has served any contentions. The court's anticipated *Markman* hearing date (assuming no stay) of March 29 or 31, 2022 (*id.*) is approximately three months after the deadline for

institution. **4:** The E.D.Va. suit likely will involve multiple grounds of invalidity, including §§101 and 112, and unique grounds under §§102 and 103 not at issue here, enabling the court to focus its limited trial time on different invalidity defenses, if the suit is not stayed. Moreover, the Petition challenges many claims (claims 4, 6, 11, 13, and 18-19) not asserted in the Complaint in the litigation. **5:** The litigation and PTAB parties are the same. **6:** The merits of the asserted grounds in this Petition are particularly strong as shown herein, and as confirmed by the PTAB having been affirmed in finding unpatentable materially similar claims of the related '552 patent based on the same **Sourcefire** prior art relied upon herein. *See* Ex. 1043; Ex. 1042. Accordingly, the Board should not exercise its discretion to deny institution.

D. 325(d) Discretion Does Not Apply

Considering the two-part framework discussed in *Advanced Bionics, LLC v. Med-El Elektromedizinische Gerate GMBH*, IPR2019-01469, Pap. 6, *8-9, the Board should not exercise its §325(d) discretion to deny institution. The grounds raised by this Petition are not the same or substantially the same as the art and arguments raised during prosecution, and to the extent there is overlap, the Examiner erred in a manner material to the patentability of the Challenged Claims.

First, the grounds raised by this Petition are not the same or substantially the same as the art and arguments raised during prosecution (step one of *Advanced*

Bionics). **Sourcefire** and IPR papers addressing the related '552 and '193 patents based on **Sourcefire** were cited in an IDS (Ex. 1002, 1313),⁵ but were not substantively discussed or applied in a rejection during prosecution. *Vizio, Inc. v. Nichia Corp.*, IPR2017-00551, Pap. 9, *7-8 (granting institution and rejecting 325(d) argument where no evidence that references cited in IDS were applied against the challenged claims or that the Examiner considered particular disclosures cited by the petitioner); *Microsoft Corp. v. Parallel Networks, LLC*, IPR2015-00486, Pap. 10, *14-15 (same). Nor is **Sourcefire** cumulative of the references considered during prosecution. For example, **Sourcefire** teaches the two-stage packet filtering process that the Examiner found missing from the prior art. See §§V.B, VIII.A. Necessarily, the Office also has not previously considered the expert testimony submitted herewith with regard these teachings.

Second, even if the art and arguments raised in the IPR petitions against the related '552 and/or '193 patents were substantially the same, the Examiner erred in a manner material to the patentability of the Challenged Claims (step two of

⁵ Applicant cited the petition, expert declaration, and institution decision in IPR2018-01436 ('552 patent) and the petition in IPR2018-01559 ('193 patent). Ex. 1002, 1314, 1438-39. The Board's final written decision invalidating all claims of the '552 patent and the Federal Circuit's affirmance were not filed.

Advanced Bionics). As an initial matter, where the “Examiner did not expressly consider” **Sourcefire**, it is difficult, if not impossible, to explain “why the Examiner allowed the claims” or “how the Examiner might have considered the arguments presented in the Petition.” *Bowtech, Inc. v. MCP IP, LLC*, IPR2019-00379, Pap. 14, *20 (declining to exercise §325(d) discretion).

Nonetheless, to the extent that Patent Owner points to the Examiner’s initials on IDSs as evidence that the Examiner considered teachings and arguments based on **Sourcefire**, the Examiner erred in a manner material to the Challenged Claims’ patentability by failing to understand **Sourcefire** and failing to apply it in a rejection. The present Petition challenges claims that recite subject matter overlapping with claims 1-21 of the ’552 patent, and at the time of the ’343 patent’s allowance, the Board had already instituted IPR against the ’552 patent based on **Sourcefire**. The Examiner erred in a manner material to the Challenged Claims by failing to identify or discuss, let alone sufficiently consider, the Board’s analysis of **Sourcefire**, which explained how multiple limitations of the Challenged Claims are met. *Ex. 1041*, 20-34; *SK Hynix v. Netlist*, IPR2020-01042, Pap. 14, *20-21 (rejecting 325(d) argument under the second part of *Advanced Bionics* where the Examiner failed to sufficiently consider a Board decision that was cited during prosecution); *Amazon v. M2M*, IPR2019-01205, Pap. 14, *12-16 (declining to exercise §325(d) discretion where “prosecution history...devoid of an

explanation about the relevance of [related IPR filings cited in IDS],” the two IPRs “present materially similar issues,” and Federal Circuit set to rule on the related IPR); *Versa Prods v. Varidesk, LLC*, IPR2020-00387, Pap. 13, *15-18.

The Examiner’s error is confirmed by the Board’s subsequent determination that *all* challenged claims of the ’552 patent are unpatentable over **Sourcefire** in a decision that has been affirmed on appeal by the Federal Circuit. Ex. 1042, 70; Ex. 1043.⁶ For example, the Board found that **Sourcefire** discloses a computing device with a processor and memory (Ex. 1042, 36-37), receiving a plurality of packets (*id.*, 37-38), the “determination” limitations (*id.*, 48), and “applying an operator in two-stage packet filtering” (*id.*, 52). The Federal Circuit affirmed the Board’s decision, finding that “[s]ubstantial evidence thus supports the Board’s finding that Sourcefire teaches the ‘determination’ limitations of the patent claims” and “the operator.” Ex. 1043, 8-9. Like the ’552 patent, the Challenged Claims here recite applying an operator in two-stage packet filtering, and are unpatentable based on

⁶ In IPR2018-01559 (’193 patent), the Board declined to institute IPR for procedural reasons—the petitioner failed to construe “operator.” Ex. 1045, 6, 9-12. Thus, the Board’s decision to deny institution in IPR2018-01559 does not impact the 325(d) analysis, and does not alter the significance of the unpatentability determination in the ’552 patent IPR.

the same **Sourcefire** ground. Indeed, the Challenged Claims are *broader* than those already found unpatentable, because the '552 claims were limited to processing HTTPS packets, whereas the Challenged Claims are not.

Finally, efficiency is served by instituting and ruling on the validity of the '343 patent as the Board is already familiar with and rejected Patent Owner's arguments in the related '552 patent IPR. The Ground presented herein is not precluded under §325(d), and the Board should institute IPR.

V. BACKGROUND

A. Summary of the '343 Patent

The '343 patent is directed to systems and methods for filtering data packets leaving a network to ensure that the packets do not improperly transfer sensitive data from the network—a breach associated with a type of cyber-attack known as exfiltration. Exfiltrations are often facilitated using popular network data transfer protocols, such as the Hypertext Transfer Protocol (HTTP) used by the Web. Such attacks may appear to an insufficiently careful observer as normal network behavior. For example, older network trust models (*e.g.*, those used by firewalls that do not interpret application-layer data like HTTP) may interpret exfiltrations as trusted operations. '343, 1:28-51.

The '343 patent purports to solve this problem and detect exfiltration attempts via a Packet Security Gateway (PSG) that receives and examines packets

that are exiting a trusted network towards an external network. A PSG is configured to receive packets and apply packet filtering rules, as dictated by a Security Policy Management Server. The packet filtering rules implement “operators” that allow or block packets based on the rules. ’343, 3:62-4:14, 8:39-52.

Figure 1 of the ’343 patent (below) shows an exemplary Network Environment 100 protected by PSGs 110 and 112 and a Security Policy Management Server 114:

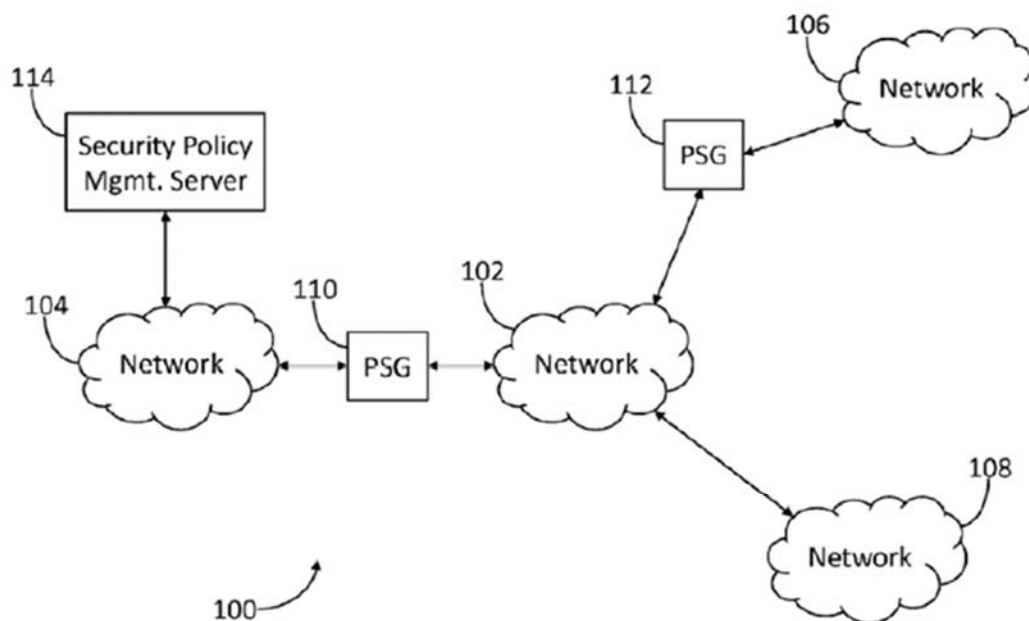
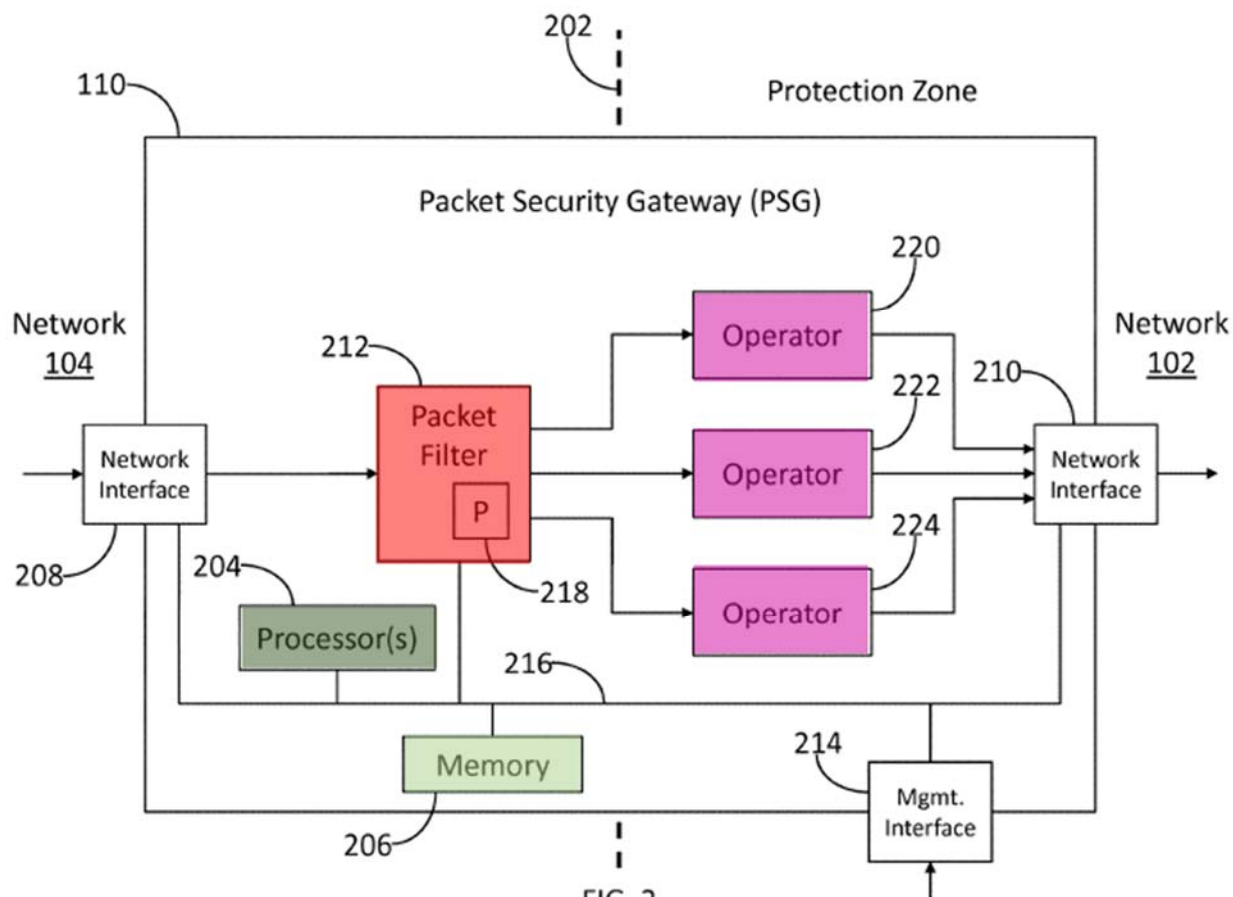


FIG. 1

Network 102, which may be the public Internet or some other large TCP/IP network, connects Network 104, Network 106, and Network 108, which may be Local Area Networks (LANs) or Wide Area Networks (WANs), such as those

operated by or associated with various organizations. Network 104 and Network 106 can be distinct LANs associated with a common enterprise X and may each form part of a protected network associated with Security Policy Management Server 114. Network 104 and Network 106 are protected by PSGs 112 and 110 respectively. Network 108 may be owned and operated by a cyber criminal organization Z, which may attempt to attack or collect data from one or more of the networks associated with enterprise X (*e.g.*, Network 104 or Network 106). In this case, PSGs 112 and 110 may protect Network 104 and Network 106 from one or more cyber attacks initiated from Network 108. '343, 3:31-4:41.

Figure 2 of the '343 patent, which has been color annotated below, shows an embodiment of PSG 110:



PSG 110 may include Network Interface 208 and Network Interface 210, which connect PSG 110 to the respective networks, and a Management Interface 214, which connects to Security Policy Management Server 114. PSG 110 may also include one or more **Processors 204** and **Memory 206**. The **Memory 206** may include program modules which, when executed by the **Processor(s) 204**, may configure the PSG 110 to enforce network policies that perform the '343 claim limitations. These policies are implemented by **Packet Filter 212** on packets that arrive at the PSG 110 via Network Interface 208. This filter will examine the packets and then forward the packets to one or more **Operators 220, 222, and 224**

based on the implemented policies and corresponding rules. An **Operator** may allow traffic, block traffic, or perform an additional function. See §VII.A (discussing claim construction for the term “operator”). If a packet is allowed to continue towards its destination, a Network Interface 210 connects the packet to its destination via Network 102. ’343, 4:59-5:49.

The PSG may perform a two-stage filtering process on the received packets.

Figure 4 (color annotated) illustrates this process:

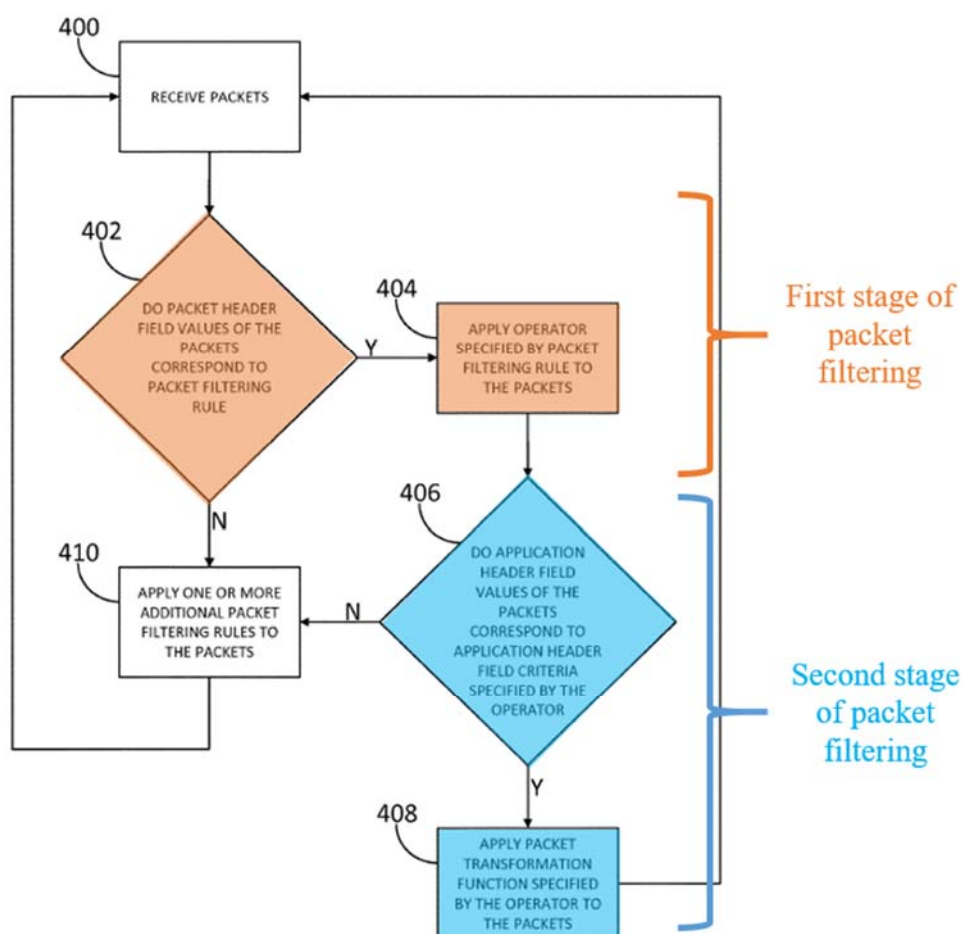


FIG. 4

Packets are received at step 400, and enter **the first filtering stage at Step 402**, where **a determination is made as to whether a portion of the received packets have packet header field values (e.g., one or more data section protocols, source IP addresses, source ports, destination IP addresses, or destination ports) that correspond to a packet filtering rule** provided by the security policy management server. If it is determined that one or more of the portion of received packets have a packet header field value corresponding to a packet filtering rule, **an operator specified by that first stage packet filtering rule is applied to the portion of received packets in Step 404**. In a one-stage filtering process, the operator would simply block or allow the packets based on the first determination described above. In the two-stage filtering process of Figure 4, however, the operator may specify one or more application-header-field criteria and a packet transformation function. Thus, **in Step 406, a determination is made as to whether one or more application header field values of one or more of the packets—i.e., the packets that entered the second stage—correspond to one or more application header field criteria** specified by the operator from Step 404. If it is determined that one or more of the portion of received packets have application header field values corresponding to one or more application header field criteria, **a packet transformation function is applied to one or more of the packets at Step 408**. A packet transformation function may allow a packet to

continue to its destination or block a packet from continuing towards its destination. '343, 2:24-39, 8:39-51, 9:11-10:47.

Figure 3 (color annotated) shows a sample set of packet filtering rules that may comprise a dynamic security policy that is dictated by a Security Policy Management Server and enforced by one or more PSGs:

218

Five-tuple

Rule #	IP Protocol	Source IP Address	Source Port	Destination IP Address	Destination Port	Operator
1 (302)	TCP	140.210.*	*	140.212.*	22	ALLOW
2 (304)	TCP	140.210.*	*	140.212.*	25	ALLOW
3 (306)	TCP	140.210.*	*	140.212.*	110	ALLOW
4 (308)	TCP	140.210.*	*	140.212.*	143	ALLOW
5 (310)	TCP	140.210.*	*	140.212.*	443	REQUIRE-TLS-1.1-1.2
6 (312)	TCP	140.210.*	*	214.*	80	HTTP-EXFIL
7 (314)	*	*	*	*	*	BLOCK

Each a two-stage packet filtering rule

FIG. 3

The **five-tuple** provides examples of the types of values available in the packet header and used as criteria for packet-filtering rules in the **first stage of filtering** for an IP packet, including IP protocol, source IP addresses, source ports, destination IP addresses, and destination ports. For example, rule 1 (302) specifies that IP packets containing one or more TCP packets, originating from an IP address that begins with 140.210, has any source port, destined for an IP address that begins with 140.212, and destined for port 22 (*e.g.*, packets associated with the

Secure Shell (SSH) protocol) should have an ALLOW operator applied to them.

This is an example of a one-stage filtering process. '343, 5:50-6:24.

Rules 5 (310) and 6 (312) are examples of two-stage packet filtering rules. The **first stage** involves an evaluation of the received packets against the criteria specified in the five-tuple, and if they correspond, the **second stage** involves an evaluation of the corresponding packets against the additional criteria specified by the operator. For example, Rule 5 (310) specifies that in **the first stage of filtering**, IP packets containing one or more TCP packets, originating from an IP address that begins with 140.210, has any source port, destined for an IP address that begins with 140.212, and destined for port 443 (*e.g.*, packets associated with the port for Hypertext Transfer Protocol Secure (HTTPS) protocol) should have a REQUIRE-TLS 1.1-1.2 Operator applied to them. Thus, in the **second stage of filtering**, the REQUIRE-TLS 1.1-1.2 Operator checks the application header values of the corresponding packets for a specified Transport Layer Security (TLS) protocol (*e.g.*, only packets matching all the packet header criteria and having application header values that indicate HTTPS sessions using TLS versions 1.1 or 1.2 will be allowed to continue). The rule allows only versions 1.1 or 1.2 of the TLS protocol “because the popular TLS version 1.0 protocol has a known security vulnerability that attackers may exploit to decrypt HTTPS sessions.” '343, 6:25-33, 6:40-7:23.

As another example, Rule 6 (312) specifies in the **first stage of filtering** that IP packets containing one or more TCP packets, originating from an IP address that begins with 140.210, has any source port, destined for an IP address that begins with 214, and destined for port 80 (*e.g.*, associated with the port for Hypertext Transfer Protocol (HTTP) protocol) should have a HTTP-EXFIL Operator applied to them. Thus, in the **second stage of filtering** applied to HTTP packets determined by the first stage of filtering, the HTTP-EXFIL Operator allows HTTP packets containing a GET method (*e.g.*, a read-only method), but blocks HTTP packets containing other HTTP methods (*e.g.*, read-write methods like PUT, POST, CONNECT, etc.). The rule blocks HTTP PUT or POST methods because they are often used to exfiltrate sensitive data. '343, 6:25-33, 6:40-7:23.

Thus, the '343 patent describes a filtering process that includes a second stage of filtering, for example to decide whether to allow or block packets using application header criteria such as (1) encryption protocol (*e.g.*, TLS versions) or (2) HTTP methods (*e.g.*, GET versus PUT, POST, CONNECT).

Each of the '343 patent's Challenged Claims is directed to an implementation of a two-stage packet filtering process as described above. Independent Claim 1 is directed to a method; independent Claim 8 is directed to a corresponding apparatus; and independent Claim 15 is directed to a corresponding computer program. Therefore, Claim 1 is representative of the independent claims.

Akl ¶¶75-84.

B. Prosecution History of the '343 Patent

The '343 patent issued from U.S. Application 15/614,956 (the '343 application”), which was filed on June 6, 2017. The '343 patent is a continuation of U.S. Application 14/625,486 (now U.S. Patent 9,686,193), filed February 18, 2015, which is a continuation of U.S. Application 13/795,822 (now U.S. Patent 9,124,552), filed March 12, 2013.

On September 27, 2018, the Examiner issued an Office Action rejecting all twenty of the pending claims. Claims 1-3, 5, 8-10, 12, and 15-17 were rejected under §103(a) as unpatentable over U.S. Pub. 2011/00559916 (“Ahn”) in view of U.S. Pub. 2003/0005122 (“Freimuth”). *Id.*, 1334. The Examiner added other references for rejections of the dependent claims. *Id.*, 1343, 1346.

On March 27, 2019, Applicant responded by amending the claims, including amendments to independent claims 1, 8, and 15 to differentiate between a “first criterion” specified to examine a packet header and a “second criterion” specified to examine an “application header field value,” and to recite a “matching” packet-filtering rule, among other changes. *Id.*, 1424, 1427, 1429. Applicant argued that Ahn fails to teach or suggest a determination based on an application header field value. Applicant further argued that Freimuth fails to “teach or suggest processing packets comprising data corresponding to a first criterion specified by at least one

matching packet-filtering rule” because after the TCP handshake completes and data is received to determine the HTTP header, the kernel of Freimuth classifies the request based on the header values. *Id.*, 1421-22.

On June 3, 2019, the Examiner again rejected all pending claims. The Examiner rejected claims 1, 8, and 15 for failing to comply with the written description requirement of §112, first paragraph because the specification did not convey that the “second portion of packets is a portion of the first portion of packets.” *Id.*, 1552. The Examiner also rejected the claims under §103(a). Claims 1-3, 5, 8-10, 12, and 15-17 were rejected as unpatentable over Ahn in view of U.S. Pub. 2012/0084866 (“Stolfo”). *Id.*, 1554. The Examiner added other references for the dependent claims. *Id.*, 1562, 1565, 1567, 1569.

On November 4, 2019, Applicant filed a Request for Continued Examination and amended independent claims 1, 8, and 15 to recite that “at least one packet transformation function indicates whether each packet in the second portion of packets is allowed to continue toward its destination” and to remove the limitation that the second portion of packets is a portion of the first portion of packets. *Id.*, 1603, 1605-08. In response to the Examiner’s rejections, Applicant argued that the references do not disclose “apply[ing], to each packet in the second portion of packets that match the second criterion, at least one packet transformation function configured to prevent an exfiltration operation, wherein the at least one packet

transformation function indicates whether each packet in the second portion of packets is allowed to continue toward its destination” as recited in independent claims 1, 8, and 15. *Id.*, 1612-13. In particular, Applicant argued that Stolfo instead disclosed generating decoy traffic rather than applying a transformation to existing network traffic, and does not prevent an exfiltration operation. *Id.*, 1613.

A Notice of Allowance issued on December 12, 2019, in which the examiner pointed to Applicant’s November 4, 2019 arguments with respect to independent claims 1, 8, and 15 as a reason for allowance. *Id.*, 1639-40. Thus, the ’343 patent’s Challenged Claims were apparently allowed because the Examiner concluded that the prior art of record failed to disclose a two-stage packet filtering process. Nevertheless, as explained herein, and as the Board was affirmed in finding, this feature was taught by the prior art relied upon herein.

Akl ¶¶85-90.

VI. LEVEL OF ORDINARY SKILL IN THE ART

For purposes of this IPR, Petitioner applies the definition of a POSITA adopted by the Board in the related ’552 patent IPR, *i.e.*, someone with working knowledge of packet-switched networking, firewalls, security policies, communication protocols and layers, and the use of customized rules to address cyber-attacks, a bachelor’s degree in computer science, computer engineering, or an equivalent, and four years of industry experience. Ex. 1042, 12-13. Lack of

work experience can be remedied by additional education, and vice versa. *Id.*; Akl ¶¶18-20.

VII. CLAIM CONSTRUCTION

Claims subject to IPR are to be construed using the *Phillips* standard. §42.100(b); *Phillips v. AWH Corp.*, 415 F.3d 1303, 1313 (Fed. Cir. 2005). Only terms necessary to resolve the controversy need to be construed. *Nidec Motor Corp. v. Zhongshan Broad Ocean Motor Co.*, 868 F.3d 1013, 1017 (Fed. Cir. 2017).⁷ Petitioner addresses one claim term below.

A. “operator”

Claim 1 recites “one or more operators specified by the at least one matching packet-filtering rule.” ’343, 11:42-44. In computer science, the term “operator” is used to refer to characters or symbols that determine an action or process to be performed. Akl ¶92. In the context of the ’343 patent, an operator represents the action or process applied to packets. The action or process may involve, for example, a programmed “packet transformation function” such as “an ALLOW packet transformation function.” ’343, 2:24-58, 5:38-40, 7:41-48, 9:53-61. Some

⁷ This claim construction analysis is not a concession as to the scope of any claim term in litigation, or a waiver of any argument in any proceeding that claim terms are indefinite, invalid, or unpatentable.

operators may specify “application header field criteria” that condition the packet transformation function; examples of these operators are characterized as “additional operators” in the specification. *Id.*, 2:28-34. The ’343 patent does not require operators to follow any particular format.

As discussed further below, the ’343 claims refer to operators that specify application header field criteria. For example, claim 1 recites, in limitation [1.c], “applying ... one or more operators specified by the at least one matching packet-filtering rule,” and recites in limitation [1.d], “determining, based on an application header field value, the second portion of packets based on whether the first portion of packets comprises data corresponding to second criterion specified by one or more operators specified by the at least one matching packet-filtering rule.” ’343, 11:41-57. Thus, the term “operator” has its plain and ordinary meaning. As recited in the claim, and like the additional operator described in the specification, the claimed “operator” specifies one or more application-header-field-value criteria, but otherwise is not limited to any particular exemplary embodiment of the additional operator described the specification.

Akl ¶¶91-93.

VIII. GROUNDS OF UNPATENTABILITY

Although the ’343 patent purports to have invented a two-stage packet filtering process, such methods were known prior to the earliest possible priority

date of the '343 patent. As demonstrated below, the prior art references render the Challenged Claims unpatentable. Akl ¶¶94-148.

A. Ground 1: Claims 1-6, 8-13, and 15-19 Are Rendered Obvious Over Sourcefire

1. Overview of Sourcefire

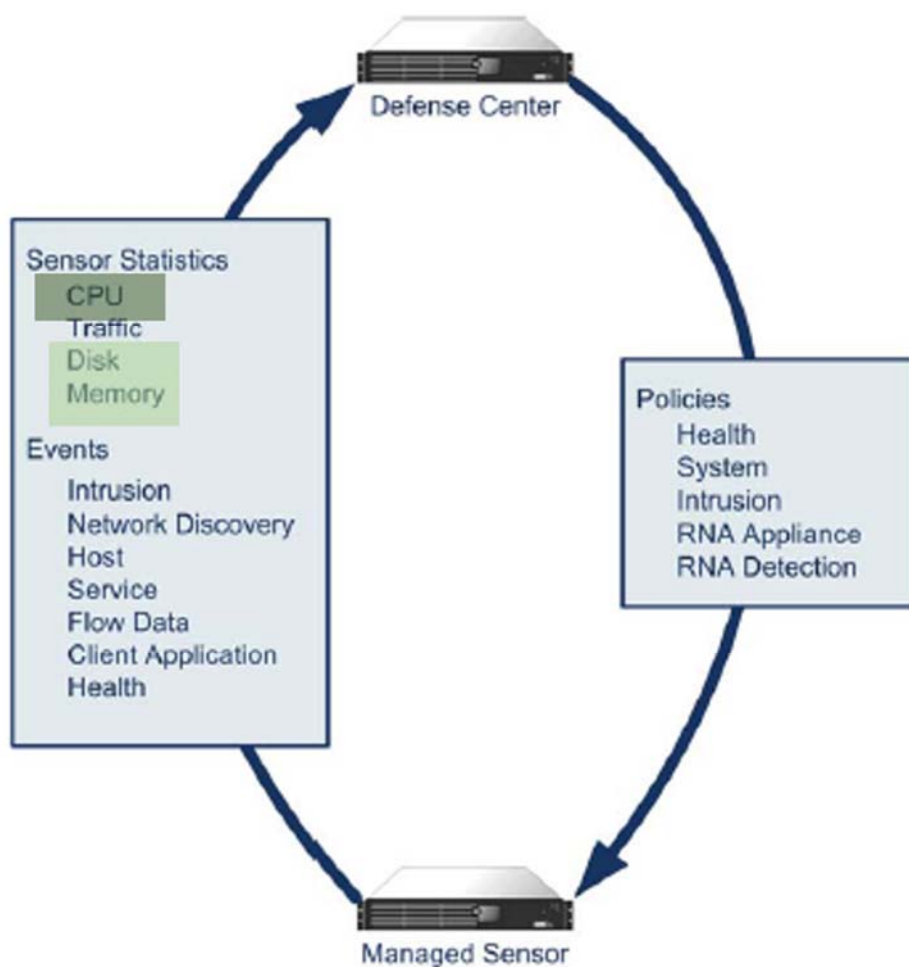
The Sourcefire 3D System User Guide (Version 4.10) (“**Sourcefire**”) is a user manual for the Sourcefire “3D System.” The 3D System provides “real-time network intelligence for real-time network defense.” Sourcefire, 32.

3D Sensors are packet-filtering devices that “aggregate network and user intelligence while defending the network against internal and external threats.” 3D Sensors run any combination of three major software components: Intrusion Prevention System (“IPS”), Real-time Network Awareness (“RNA”), and Real-time User Awareness (“RUA”). Sourcefire, 33-34.

IPS utilizes a rules-based detection engine that allows monitoring of the network for attacks that might affect the availability, integrity, or confidentiality of hosts on the network. 3D Sensors that use IPS include a set of pre-developed intrusion rules that detect common attacks. The user can also create custom intrusion rules. If 3D Sensors are deployed inline on the network to create an inline detection engine, the 3D Sensors can be configured to drop or replace packets based on the intrusion rules. Sourcefire, 33-35.

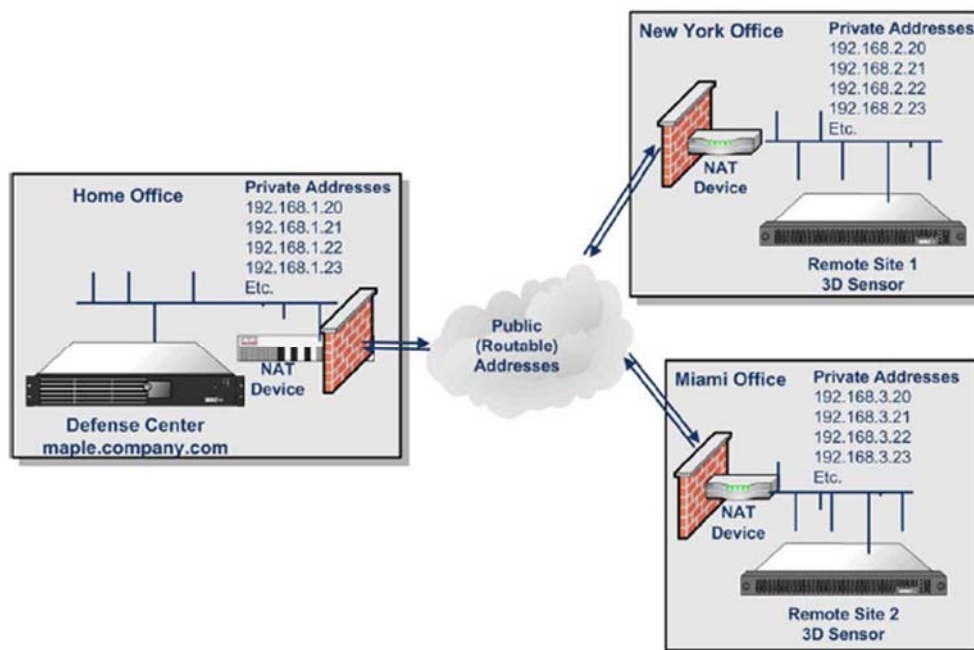
Each 3D Sensor includes a **processor** (CPU), **memory**, and **disk storage**.

3D Sensors can be managed by a Defense Center, in which case these sensors will send updates to the Defense Center, and the Defense Center in turn pushes health, system, and detection policies to the 3D Sensors. The below figure illustrates an exemplary system containing a 3D Sensor and a Defense Center.



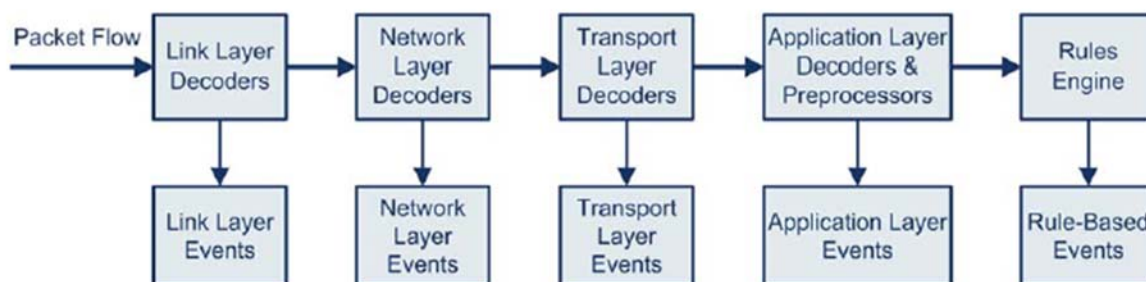
Sourcefire, 107.

3D Sensors are placed at a boundary of protected networks, as shown in the below figure.



Sourcefire, 118.

3D Sensors have three different configurations: passive, inline, or inline with fail-open. In inline deployments, a sensor's inline rules can be configured "to drop suspicious packets." Sourcefire, 70, 222, 253, 254, 262, 263, 264. The following figure illustrates how traffic is analyzed by the 3D Sensor:



Sourcefire, 256.

The packets are decoded by packet decoders which convert the packet headers and payloads into a format that can be easily used by the preprocessors and rules engine. These decoders include “a link layer decoder such as the Ethernet II decoder, a network layer decoder such as the IP decoder, a transport layer decoder such as the TCP decoder, [and] an application layer decoder.” After these packets are decoded, they are sent to “Sourcefire preprocessors [which] normalize traffic and help identify network layer and transport layer protocol anomalies.” An example preprocessor is an application layer preprocessor such as the HTTP Inspect preprocessor.

After preprocessing, packets in the 3D Sensor are sent to a Rules Engine. “As packets arrive at the rules engine, it selects the appropriate rule subsets to apply to each packet.” These rules can “restrict packet inspection to the packets originating from specific IP addresses or those destined to a specific IP address.” Preprocessors and the rules engine can also be programmed to follow drop rules. Sourcefire, 256-257, 259-260, 484, 766.

The 3D Sensors are sold with pre-programmed rules that the user can select from. A user can also write custom rules, either by modifying pre-programmed rules or by writing new rules altogether. **Sourcefire** encourages this, explaining that “the most successful rules target traffic that may attempt to exploit known

vulnerabilities rather than specific known exploits.” Sourcefire, 762. Rules can be applied on a packet-by-packet basis. Sourcefire, 259.

Sourcefire discloses that intrusion rules have two logical sections: the rule header and the rule options. The rule header, which has 5-tuple values associated with it, contains:

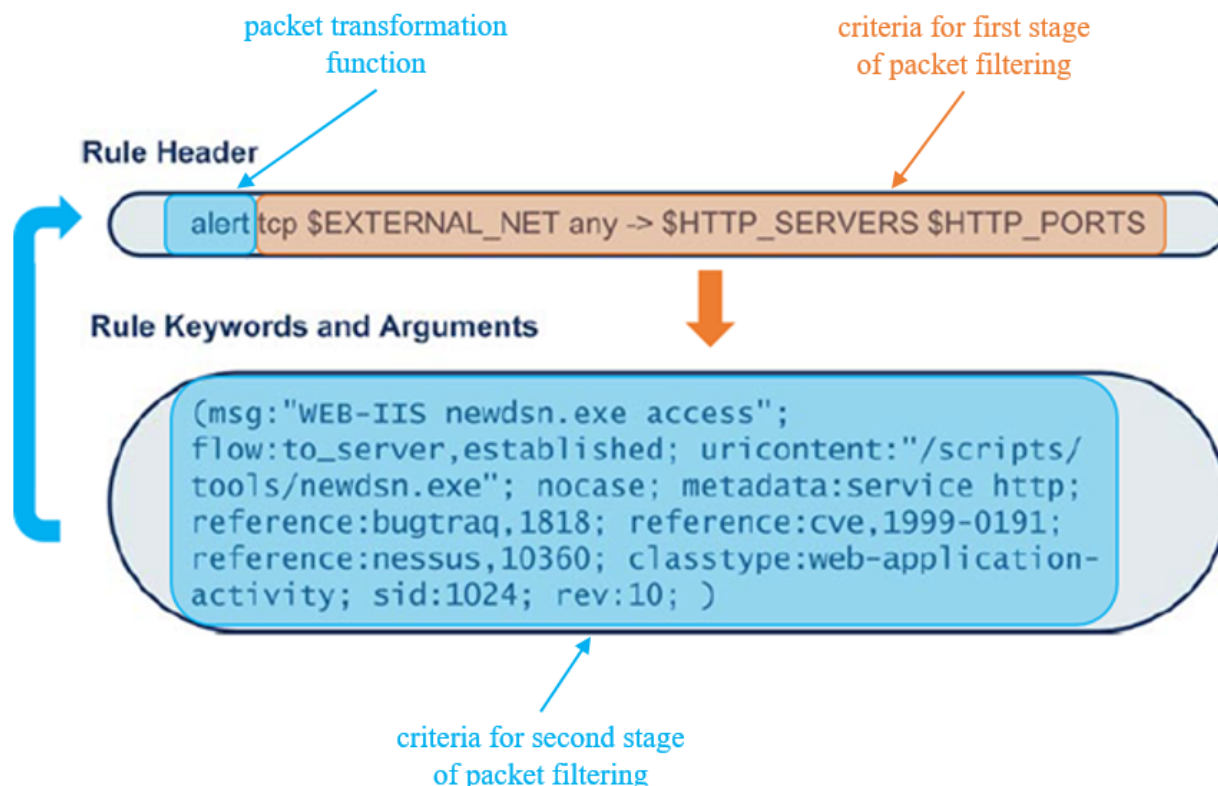
- The 5-tuple:
 - Protocol
 - Source and destination IP addresses
 - Source and destination ports
- The rule’s action or type (e.g., alert and allow, drop, ignore and allow).

The rule options contain:

- Event messages
- Keywords and their parameters and arguments
- Patterns that a packet’s payload must match to trigger the rule
- Specification of which parts of the packet the rules engine should inspect

Sourcefire, 256.

Sourcefire provides an example intrusion rule that shows these different parts of an intrusion rule (*i.e.*, a “rule header” which includes parameter and arguments and “rule options” which include keywords and arguments):



Sourcefire, 763 (annotated with labels and arrows to illustrate the rule flow). Based on this rule's structure, this rule is used for a two-stage filtering process, as indicated by the annotations in the above figure.

For example, **Sourcefire** discloses **filtering packets based on packet header information (i.e., a first stage of filtering using first criteria)**. This stage of processing occurs at the network layer decoder (which looks at IP address and protocol) and the transport layer decoder (which looks at ports). This packet header information is compared against the 5-tuple values in the rule header (i.e., the protocol, the source and destination IP addresses, and the source and destination

ports), just like Stage 1 of filtering in the '343 patent. Sourcefire, 256-259, 764-769.

Sourcefire further discloses **applying, to packets that match the first criteria, “rule keywords and arguments” from the “rule options” (i.e., second criteria) and applying a rule action (i.e., a packet transformation function) to packets that match the rule keywords and arguments (i.e., a second stage of filtering using second criteria)**. A user can specify the behavior of a rule by combining keywords. Keywords and their associated values (called arguments) dictate how the system evaluates packets and packet-related values that the rules engine tests. The Sourcefire 3D System supports keywords that allow the user to perform inspection functions, such as content matching, protocol-specific pattern matching, and state-specific matching. Any number of compatible keywords can be combined to create highly specific rules. The rule also specifies a “rule action” that will pass, alert (and pass), or drop packets that meet the criteria indicated by the rule keywords and arguments specified by the rule. Sourcefire, 297, 359, 761, 765, 769. The rule keywords/arguments are an “operator” as that term is used in the Challenged Claims.

As one example of rule keywords for second-stage filtering, **Sourcefire** discloses SSL rule keywords that invoke the Secure Sockets Layer (SSL) preprocessor to **parse SSL/TLS traffic for encrypted sessions and match**

against SSL/TLS version information. Thus, a user could protect a network against a vulnerability associated with a particular encryption protocol version (*e.g.*, TLS Version 1.0) by writing a rule using the `ssl_version` keyword that invokes the SSL preprocessor to identify packets that use that protocol version, which had known vulnerabilities.

As another example, **Sourcefire** discloses that HTTP content keywords can be used to write preprocessor and intrusion rules for another application-layer preprocessor, called the HTTPS inspect preprocessor, and for the rules engine, to **search for matched application-layer header content in packets using known HTTP methods “such as GET, PUT, [and] CONNECT.”** Sourcefire, 491,761, 785-786, 807.

Thus, **Sourcefire** describes **filtering packets based on packet header information including the 5-tuple (first criteria)**, just like the Stage I evaluation described in the '343 patent. Every intrusion rule is qualified by the 5-tuple and, for a given 3D Sensor with IPS, this Stage I evaluation is performed as a packet arrives at the network layer decoder looking at IP address and protocol, and the transport layer decoder looking at ports. Sourcefire, 256-259, 764-769. **Sourcefire** further describes how a user could design various rules to **pass or drop packets (apply a packet transformation function) based on application-layer header information such as SSL/TLS version or HTTP methods (second criteria)**, just

like the Stage II evaluation described in the '343 patent. Sourcefire, 256-260, 297, 359, 761, 765.

Akl ¶¶94-110.

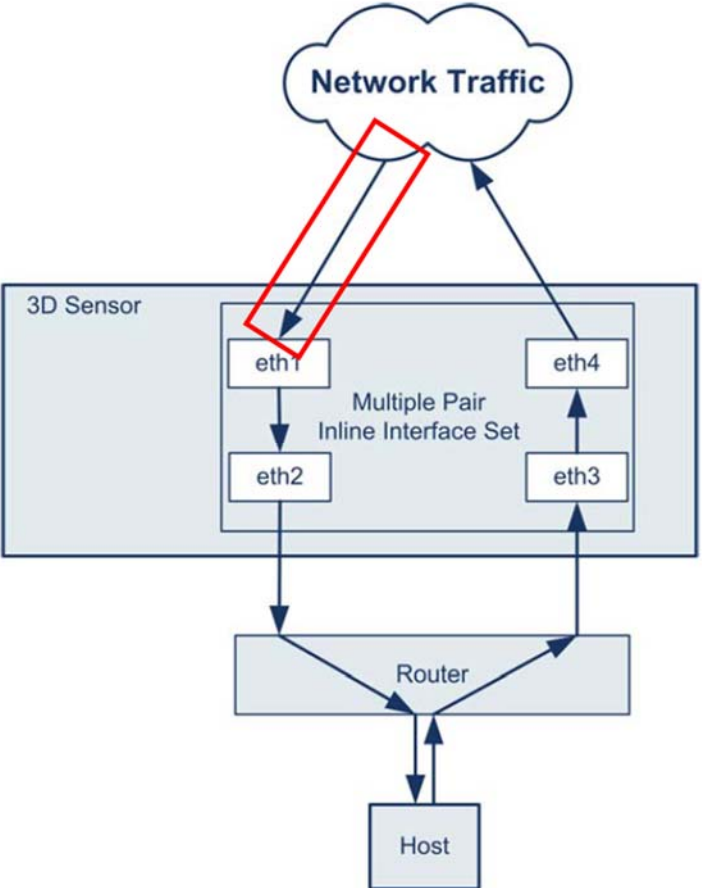
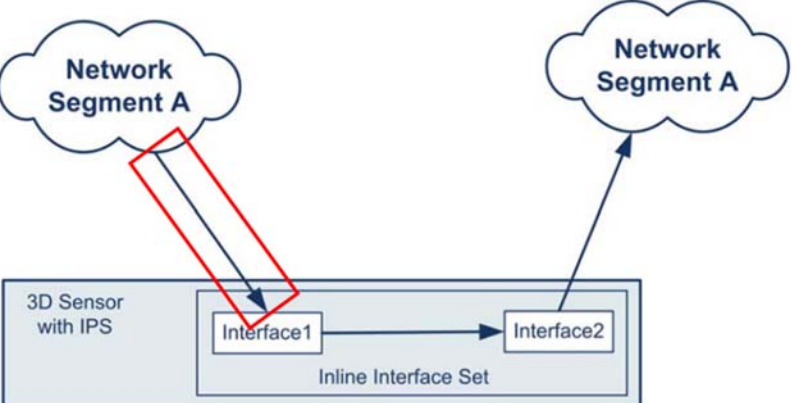
2. Claim Chart

Claim Language	Sourcefire
[1.pre] A method comprising: [1.a] receiving, by a computing system comprising memory and at least one processor, a plurality of packets, w[herein the plurality of packets comprises a first portion of packets and a second portion of packets; [8.pre] An apparatus comprising: [8.a] at least one processor; and [8.b] memory storing	Sourcefire discloses [1.pre] a method comprising: [1.a] receiving, by a computing system (e.g., “3D Sensor”) comprising memory (e.g., comprising “memory,” “disk”) and at least one processor (e.g., comprising a “CPU”), a plurality of packets, wherein the plurality of packets comprises a first portion of packets and a second portion of packets (e.g., receiving “packets” comprising first and second portions). Sourcefire discloses [8.pre] an apparatus (e.g., “3D Sensor”) comprising: [8.a] at least one processor (e.g., “CPU”); and [8.b] memory (e.g., “memory,” “disk”) storing instructions that, when executed by the at least one processor (e.g., executed by the “processor,” “CPU”), cause the apparatus to: [8.c] receive a plurality of packets (e.g., receiving “packets”); Sourcefire discloses [15.pre] one or more non-transitory computer-readable media (e.g., “memory,” “disk”) comprising instructions that, when executed by one or more computing devices (e.g., executed by the “CPU” of the “3D Sensor”), cause the one or more computing devices to: [15.a] receive a plurality of packets (e.g., receiving “packets”); The Board in the IPR for the related '552 patent found that Sourcefire teaches a computing device with a processor and memory, which receives a plurality of packets. Ex. 1042, 36-

Claim Language	Sourcefire
instructions that, when executed by the at least one processor, cause the apparatus to: [8.c] receive a plurality of packets; [15.pre] One or more non-transitory computer-readable media comprising instructions that, when executed by one or more computing devices, cause the one or more computing devices to: [15.a] receive a plurality of packets;	38. Patent Owner is therefore collaterally estopped here from relitigating this issue.⁸ For example, Sourcefire discloses a computing system referred to as a “3D Sensor” which comprises memory, such as RAM and disk storage, and a processor. A POSITA would have understood that the software that runs on the 3D sensor to implement custom intrusion rules is stored in memory (<i>e.g.</i>, RAM or disk) and executes on the CPU. Akl ¶117. • 90: “The System Load widget shows the <i>CPU usage (for each CPU), memory (RAM) usage...</i>” • 204: “The maximum number of detection engines per sensor is two times the number of CPUs (for example, if a <i>sensor has 24 CPUs</i>, you can create up to 48 detection engines).” • 107:

⁸ A complete comparison of the Challenged Claims to the claims of the ’552 patent (Ex. 1040) is attached as Ex. 1044.

Claim Language	Sourcefire
	Sourcefire discloses receiving by the computing system a plurality of packets. For example, the 3D Sensor receives internet traffic and captures packets within that traffic for examination. 118:

Claim Language	Sourcefire
	235:  264: 

Claim Language	Sourcefire
	34: “By placing 3D Sensors with IPS on key network segments, you can <i>examine the packets</i> that traverse your network for malicious activity.” 257: “Before packets can be inspected by the IPS detection engine, the <i>packets must be captured from the network</i>. The following illustration shows how the detection engine sniffs packets, then decodes them prior to any further analysis.” 257: <pre> graph TD PS[Packet Stream] --> NIC[NIC] NIC -- "Ethernet, SLIP, 802.11, etc." --> PD[Packet Decoder] subgraph "3D Sensor with IPS" NIC PD end </pre> 264: “[T]he detection engine using an inline interface set can <i>see all of the traffic that passes through the interfaces</i> in its interface set, regardless of the traffic’s destination.” - See also 33, 118, 222-230, 234-235, 253-254, 257, 262-263, 265, 435-439. A POSITA would have understood that the plurality of packets have a first portion of packets and a second portion of packets. Akl ¶117. For example, in the two-stage packet filtering process described below with respect to Elements [1.b] through [1.e], received packets are compared against criteria in the rule header, and then the packets that correspond to those criteria (<i>i.e.</i> a first portion of packets) are compared against criteria in the rule keywords and arguments to determine which packets are allowed (<i>i.e.</i> a second portion of packets).

Claim Language	Sourcefire
	Akl ¶117.
[1.b]/[8.d]/[15.b] [determining/determine], based on a packet header field value, whether [each packet of] the plurality of packets comprises data corresponding to first criterion specified by one or more packet-filtering rules;	Sourcefire discloses determining, based on a packet header field value (<i>e.g.</i>, based on values such as “protocol” or “source and destination IP address” in the packet header that are obtained by “decod[ing] through the first three TCP/IP layers”), whether each packet of the plurality of packets comprises data corresponding to first criterion (<i>e.g.</i>, “compares packets against the conditions specified in each rule” which begins with a first stage of applying criteria such as “protocol” or “source and destination IP addresses” in a 5-tuple in the “rule header”) specified by one or more packet-filtering rules (<i>e.g.</i>, “intrusion rules”). The Board in the IPR for the related ’552 patent found that Sourcefire teaches a determination of whether packets have packet-header-field values (5-tuple information) corresponding to a packet filtering rule. Ex. 1042, 38, 48. Patent Owner is therefore collaterally estopped here from relitigating this issue. Sourcefire discloses first criterion specified by packet-filtering rules. For example, Sourcefire discloses intrusion rules. Sourcefire, 428-430. Each rule has two parts: the rule header and the rule options. Each rule header includes a 5-tuple (shown in orange, below) specifying the protocol, the source and destination IP address, and the source and destination ports that the rule applies to (<i>i.e.</i>, first criteria). This is consistent with the ’343 patent’s disclosure that criteria can include data in the “form of a five-tuple” which may comprise values such as protocol type, source IP addresses, or destination IP address. ’343 patent, 5:53-64. • 763:

Claim Language	Sourcefire																								
	packet transformation function criteria for first stage of packet filtering Rule Header Rule Keywords and Arguments criteria for second stage of packet filtering 764: Type Source IP Operator Destination Port Protocol Source Port Destination IP Rule Header Values <table><thead><tr><th>Rule Header Component</th><th>Example Value</th><th>This Value...</th></tr></thead><tbody><tr><td>Action</td><td>alert</td><td>Generates an intrusion event when triggered.</td></tr><tr><td>Protocol</td><td>tcp</td><td>Tests TCP traffic only.</td></tr><tr><td>Source IP Address</td><td>\$EXTERNAL_NET</td><td>Tests traffic coming from any host that is not on your internal network.</td></tr><tr><td>Source Ports</td><td>any</td><td>Tests traffic coming from any port on the originating host.</td></tr><tr><td>Operator</td><td>-></td><td>Tests external traffic (destined for the web servers on your network).</td></tr><tr><td>Destination IP Address</td><td>\$HTTP_SERVERS</td><td>Tests traffic to be delivered to any host specified as a web server on your internal network.</td></tr><tr><td>Destination Ports</td><td>\$HTTP_PORTS</td><td>Tests traffic delivered to an HTTP port on your internal network.</td></tr></tbody></table> 761: “An intrusion rule is a specified set of keywords and arguments on a 3D Sensor with the IPS component that detects attempts to exploit vulnerabilities on your network by analyzing network traffic to check if it	Rule Header Component	Example Value	This Value...	Action	alert	Generates an intrusion event when triggered.	Protocol	tcp	Tests TCP traffic only.	Source IP Address	\$EXTERNAL_NET	Tests traffic coming from any host that is not on your internal network.	Source Ports	any	Tests traffic coming from any port on the originating host.	Operator	->	Tests external traffic (destined for the web servers on your network).	Destination IP Address	\$HTTP_SERVERS	Tests traffic to be delivered to any host specified as a web server on your internal network.	Destination Ports	\$HTTP_PORTS	Tests traffic delivered to an HTTP port on your internal network.
Rule Header Component	Example Value	This Value...																							
Action	alert	Generates an intrusion event when triggered.																							
Protocol	tcp	Tests TCP traffic only.																							
Source IP Address	\$EXTERNAL_NET	Tests traffic coming from any host that is not on your internal network.																							
Source Ports	any	Tests traffic coming from any port on the originating host.																							
Operator	->	Tests external traffic (destined for the web servers on your network).																							
Destination IP Address	\$HTTP_SERVERS	Tests traffic to be delivered to any host specified as a web server on your internal network.																							
Destination Ports	\$HTTP_PORTS	Tests traffic delivered to an HTTP port on your internal network.																							

Claim Language	Sourcefire
	matches the criteria in the rule. IPS <i>compares packets against the conditions specified in each rule</i> and, if the packet data matches all the conditions specified in a rule, the rule triggers.” • 762: “All standard text rules contain two logical sections: the rule header and the rule options. <i>The rule header contains:</i> ○ the rule’s action or type ○ <i>the protocol</i> ○ <i>the source and destination IP addresses and netmasks</i> ○ direction indicators showing the flow of traffic from source to destination ○ <i>the source and destination ports</i> • 766: “you must <i>specify the protocol of the traffic the rule inspects</i>” • 766: “You can <i>restrict packet inspection to the packets originating from specific IP addresses or those destined to a specific IP address.</i>” • 768: “You can <i>restrict packet inspection to packets originating from or destined to specific ports.</i>” • 861-62: “[Y]ou can <i>use the rule header settings to focus the rule to only match traffic using a specific protocol and traveling to or from specific IP addresses or ports.</i>” • See also 254, 257, 260, 358-361, 428-430, 485, 624-626, 645, 765-771, 869-872. Sourcefire discloses determining whether a packet header field value (<i>e.g.</i>, protocol, source and destination IP address, source and destination ports) of each packet has data corresponding to the first criterion of the packet-filtering rule

Claim Language	Sourcefire
	(i.e., corresponding to the 5-tuple values specified by the rule header). As explained above, the rule header in an intrusion rule specifies the protocol, source and destination addresses, and source and destination ports that identify the packets to be inspected. As each packet is received by the 3D Sensor, it is decoded by link layer, network layer, transport layer, and application layer decoders to extract header information before being sent to a set of preprocessors and the rules engine. Thus, a POSITA would have understood that the decoders, preprocessor and/or rules engine implementing an intrusion rule would extract the protocol, source and destination IP addresses, and source and destination ports (i.e., packet header field values) from the header for at least a portion of received packets, and compare those values to the 5-tuple values in the rule header. Akl ¶118. 260: <pre> graph LR subgraph "3D Sensor with IPS" direction TB subgraph " " direction LR PD[Packet Decoder] --> P[Preprocessors] P --> RE[Rules Engine] end ED[Event Database] PD --> ED P --> ED RE --> ED end </pre> 256: “Intrusion events can be generated by: - o a link layer decoder such as the Ethernet II decoder - o a network layer decoder such as the IP decoder - o a transport layer decoder such as the TCP decoder - o an application layer decoder or preprocessor such as the HTTP Inspect preprocessor - o the rules engine”

Claim Language	Sourcefire										
	621: “Before sending captured packets to a preprocessor, the detection engine first sends the packets to the packet decoder. The packet decoder converts packet headers and payloads into a format that can be easily used by the preprocessors and the rules engine. Each stack layer is decoded in turn, beginning with the data link layer and continuing through the network and transport layers.” 258: <table> <tr> <th colspan="2">Decoded Packets</th></tr> <tr> <th>In this TCP/IP layer...</th><th>These packets are decoded...</th></tr> <tr> <td>Data Link</td><td> - raw packets - loopback interface - Ethernet - Token Ring - Fiber Distributed Data Interface (FDDI) - Linux SLL (cooked mode) - IEEE 802.11 - Point-to-Point Protocol (PPP) - Point-to-Point Protocol over Ethernet (PPPOE) - Serial Line Internet Protocol (SLIP) - ISDN for Linux (I4L) - Address Resolution Protocol (ARP) - Extensible Authentication Protocol (EAP) - EAP over LAN (EAPOL) </td></tr> <tr> <td>Network</td><td> - Internet Protocol (IP) - Internet Control Message Protocol (ICMP) </td></tr> <tr> <td>Transport</td><td> - Transmission Control Protocol (TCP) - User Datagram Protocol (UDP) </td></tr> </table> 258-259: “After the packets are decoded through the first three TCP/IP layers, they are sent to preprocessors, which normalize traffic at the application layer and detect protocol anomalies. After the packets have passed through the preprocessors, they are sent to the rules engine. The rules engine inspects the packet headers and payloads to determine whether they trigger any of the ... rules.” 602: “Before packets are sent to preprocessors, the packet decoder converts packet headers and payloads	Decoded Packets		In this TCP/IP layer...	These packets are decoded...	Data Link	- raw packets - loopback interface - Ethernet - Token Ring - Fiber Distributed Data Interface (FDDI) - Linux SLL (cooked mode) - IEEE 802.11 - Point-to-Point Protocol (PPP) - Point-to-Point Protocol over Ethernet (PPPOE) - Serial Line Internet Protocol (SLIP) - ISDN for Linux (I4L) - Address Resolution Protocol (ARP) - Extensible Authentication Protocol (EAP) - EAP over LAN (EAPOL)	Network	- Internet Protocol (IP) - Internet Control Message Protocol (ICMP)	Transport	- Transmission Control Protocol (TCP) - User Datagram Protocol (UDP)
Decoded Packets											
In this TCP/IP layer...	These packets are decoded...										
Data Link	- raw packets - loopback interface - Ethernet - Token Ring - Fiber Distributed Data Interface (FDDI) - Linux SLL (cooked mode) - IEEE 802.11 - Point-to-Point Protocol (PPP) - Point-to-Point Protocol over Ethernet (PPPOE) - Serial Line Internet Protocol (SLIP) - ISDN for Linux (I4L) - Address Resolution Protocol (ARP) - Extensible Authentication Protocol (EAP) - EAP over LAN (EAPOL)										
Network	- Internet Protocol (IP) - Internet Control Message Protocol (ICMP)										
Transport	- Transmission Control Protocol (TCP) - User Datagram Protocol (UDP)										

Claim Language	Sourcefire
	<i>into a format that can be easily used by the preprocessors and the rules engine</i> and detects various anomalous behaviors in packet headers.” • See also 35, 276-277, 428-430, 465, 484, 488-89, 491, 575, 596-601, 608, 625-627, 761-768, 861-864, 2084, 2088-2089, 2093. Akl ¶118.
[1.c]/[8.e]/[15.c] responsive to a determination [by the computing system] that a packet header field value of [the/a] first portion of packets comprises data corresponding to the first criterion specified by at least one matching packet-filtering rule, [applying/apply], [by the computing system and] to each packet in the first portion of packets, one or more operators specified by the at least one	Sourcefire discloses responsive to a determination (e.g., “if the packet data matches all the conditions specified in a rule, the rule triggers”) by the computing system that a packet header field value of the first portion of packets comprises data corresponding to the first criterion specified by at least one matching packet-filtering rule (e.g., see [1.b]), applying, by the computing system and to each packet in the first portion of packets, one or more operators (e.g., applying “rule keywords and arguments” and the “rule action”) specified by the at least one matching packet-filtering rule (e.g., specified by “intrusion rules”). The Board in the IPR for the related ’552 patent IPR found that Sourcefire teaches a determination of whether packets have packet-header-field values (5-tuple information) corresponding to a packet filtering rule, and applying an “operator” to packets that meet that criteria. Ex. 1042, 38, 51-53 (“[W]e agree with Petitioner that...Sourcefire discloses determining whether to apply an operator in a two-stage packet filtering operation.”). The Federal Circuit affirmed the Board’s decision, including its findings with respect to the “determination” limitations and the “operator.” Ex. 1043, 8-9 (“The language of the claims and of Sourcefire provide substantial evidence for the Board’s finding that Sourcefire teaches the operator in the ’552 patent’s claims.”). Patent Owner is therefore collaterally estopped here from relitigating this issue.

Claim Language	Sourcefire
matching packet-filtering rule;	As discussed above with respect to [1.b], Sourcefire discloses comparing packet header information, <i>e.g.</i>, 5-tuple values as part of the first stage of packet filtering (shown in orange, below), which determines whether to apply the “rule keywords and arguments” to a given packet. The keywords and arguments are applied to determine whether to apply the “rule action,” such as “pass” or “drop,” to the packet being inspected. 763: 769 (“Using the rules language, you can specify the behavior of a rule by combining keywords. Keywords and their associated values (called arguments) dictate how the system evaluates packets and packet-related values that the rules engine tests. You can combine any number of compatible keywords to create highly specific rules.”) 761: “An intrusion rule is a specified set of keywords and arguments on a 3D Sensor with the IPS component that detects attempts to exploit vulnerabilities on your network by analyzing network traffic to check if it matches the criteria in the <i>rule</i>. IPS compares packets against the conditions specified in each rule and, if the packet data matches all the conditions specified in a rule, the rule triggers.”

Claim Language	Sourcefire
	• 765: “Specifying Rule Actions ... Each rule header includes a parameter that <i>specifies the action the system takes when a packet triggers a rule.</i> ... Rules with the action set to pass do not generate an event against, or log the details of, the packet that triggered the rule. ... In an inline deployment, rules with the rule state set to Drop and Generate Events generate an intrusion event against the packet that triggered the rule.” • See also 259, 358-359, 399, 403-406, 435-439, 464-466, 599, 762, 765-769. Sourcefire thus discloses that the 3D Sensor applies “keywords and arguments” to each packet in the first portion of packets (<i>i.e.</i>, to packets that match the 5-tuple in the rule header), responsive to a determination by the 3D Sensor that a packet header field value (<i>e.g.</i>, protocol, source and destination IP address, source and destination ports) of the first portion of packets comprises data corresponding to the first criterion specified by at least one matching packet-filtering rule (<i>i.e.</i>, corresponding to the 5-tuple values specified by the rule header). Only if the first stage criterion is met (<i>e.g.</i>, 5-tuple values) are remaining criteria (<i>i.e.</i>, rule keywords and arguments) in the rule tested. Then the rule’s action or type (<i>e.g.</i>, allow, drop) is applied only to the packets that satisfy the keywords and arguments. Sourcefire’s teaching of intrusion rules that apply “keywords and arguments” is a teaching of applying an “operator” as that term is used in the Challenged Claims. As explained in Section VII.A, an “operator” as used in the Challenged Claims specifies one or more application-header-field-value criteria.

Claim Language	Sourcefire
	Sourcefire discloses applying an “operator” that meets this requirement, as set forth below with respect to [1.d] (requiring applying a second criteria “based on an application header field value”). Akl ¶119.
[1.d]/[8.f]/[15.d] [determining/determine], based on an application header field value, [the/a] second portion of packets based on whether the first portion of packets comprises data corresponding to second criterion specified by one or more operators specified by the at least one matching packet-filtering rule; and	Sourcefire discloses determining, based on an application header field value (e.g., “application layer protocol values,” HTTP “request method field, which identifies the action such as GET, PUT, CONNECT,” “SSL_version”), the second portion of packets (e.g., packets matching the first criterion and the second criterion) based on whether the first portion of packets (e.g., see [1.c]) comprises data corresponding to second criterion specified by one or more operators (e.g., “rule keywords and arguments”) specified by the at least one matching packet-filtering rule (e.g., specified by “intrusion rules,” see [1.c]). The Board in the IPR for the related ’552 patent IPR found that Sourcefire teaches applying an operator that specifies application-header-field values (second criteria) as part of a two-stage packet filtering process. Ex. 1042, 38, 51-53. The Federal Circuit affirmed. Ex. 1043, 8-9. Patent Owner is therefore collaterally estopped here from relitigating this issue. Sourcefire discloses at least two examples of rule keywords/arguments for identifying matching packets based on an application-layer field value (e.g., rules examining HTTP methods and SSL/TLS versions). As a first example, Sourcefire discloses an HTTP Method option in the HTTP content keywords specification list for checking HTTP methods, such as GET, PUT, and CONNECT, in HTTP request packets. Information about

Claim Language	Sourcefire
	HTTP packets is decoded by the HTTP inspect preprocessor, which is an application-layer preprocessor. Akl ¶120. • 560-561: “The <i>HTTP Inspect preprocessor</i> is responsible for: ○ <i>decoding and normalizing HTTP requests</i> sent to and HTTP responses received from web servers on your network ○ ... ○ making the normalized data available for additional rule processing • 568: “HTTP Methods Specifies <i>HTTP request methods for the preprocessor to inspect in addition to GET and POST</i>. Use a comma to separate multiple values.” • 786: “HTTP content Keyword Options Select this option... <i>HTTP Method</i> To <i>search for content matches in... the request method field, which identifies the action such as GET, PUT, CONNECT</i>, and so on to take on the resource identified in the URI.” • 806: “The HTTP preprocessor must be enabled to allow processing of rules using the C, H, U, M, or P expression modifiers.” • 807:

Claim Language	Sourcefire														
	Snort-Specific Post Regular Expression Modifiers (Continued) <table> <tr> <th>Option</th><th>Description</th></tr> <tr> <td>H</td><td>Searches for the content within the header, excluding cookies, of an HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>D</td><td>Searches for the content within the header, excluding cookies, of a raw HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Raw Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>M</td><td>Searches for the content within the method field of a normalized HTTP request message decoded by the HTTP Inspect preprocessor; the method field identifies the action such as GET, PUT, CONNECT, and so on to take on the resource identified in the URI. See the content keyword HTTP Method option in the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>C</td><td>When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the normalized content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>K</td><td>When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the raw content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Raw Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>S</td><td>Searches the 3-digit status code in an HTTP response. See the content keyword HTTP Status Code option in the HTTP content Keyword Options table on page 785 for more information.</td></tr> </table> • See also 480, 484, 513, 569, 621, 806, 783-786, 826-837, 859, 880. To the extent Sourcefire does not explicitly describe a rule in which an HTTP method such as GET, PUT, or CONNECT is specifically included as a keyword, a POSITA would have found obvious a rule using the HTTP Method as part of the criteria to invoke the HTTP inspect preprocessor to identify traffic using that HTTP method (<i>e.g.</i>, PUT), and either pass or drop those packets based on the inspection. Akl ¶120. For example, Sourcefire discloses creating “your own Sourcefire	Option	Description	H	Searches for the content within the header, excluding cookies, of an HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.	D	Searches for the content within the header, excluding cookies, of a raw HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Raw Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.	M	Searches for the content within the method field of a normalized HTTP request message decoded by the HTTP Inspect preprocessor; the method field identifies the action such as GET, PUT, CONNECT, and so on to take on the resource identified in the URI. See the content keyword HTTP Method option in the HTTP content Keyword Options table on page 785 for more information.	C	When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the normalized content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.	K	When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the raw content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Raw Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.	S	Searches the 3-digit status code in an HTTP response. See the content keyword HTTP Status Code option in the HTTP content Keyword Options table on page 785 for more information.
Option	Description														
H	Searches for the content within the header, excluding cookies, of an HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.														
D	Searches for the content within the header, excluding cookies, of a raw HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Raw Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.														
M	Searches for the content within the method field of a normalized HTTP request message decoded by the HTTP Inspect preprocessor; the method field identifies the action such as GET, PUT, CONNECT, and so on to take on the resource identified in the URI. See the content keyword HTTP Method option in the HTTP content Keyword Options table on page 785 for more information.														
C	When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the normalized content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.														
K	When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the raw content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Raw Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.														
S	Searches the 3-digit status code in an HTTP response. See the content keyword HTTP Status Code option in the HTTP content Keyword Options table on page 785 for more information.														

Claim Language	Sourcefire
	3D System rules.” Sourcefire, 377, 762. Sourcefire further discloses that intrusion rules can be implemented by the HTTP inspect preprocessor and by the rules engine and provides a specific keyword option to access the HTTP method. Sourcefire, 568 (“Specifies HTTP request methods for the preprocessor to inspect in addition to GET and POST.”); <i>see also</i> 435-439, 491, 569, 785-786, 807. Additionally, a POSITA would have understood that HTTP methods such as PUT, POST, and CONNECT were well-known methods of network exfiltration, because a malicious actor can use such methods to send data out of a network. This is the reason that Sourcefire teaches incorporating keywords into packet filtering rules to identify these methods. Akl ¶120.⁹ Indeed, existing packet-filtering firewalls used application methods as part of rule criteria to block and pass packets, <i>e.g.</i>, a known “stateful inspection firewall allowed the FTP ‘GET’ command but disallowed the ‘PUT’ command” (Ex. 1008, 4:31-41), and HTTP methods were used in IDS rules at least as early as 1997 (Ex. 1028, 4-80). Akl ¶120. Thus, in view of the teachings of Sourcefire, a POSITA would have been motivated and found it obvious to write an intrusion rule in which the HTTP Method option is specifically included in a rule in order to detect packets containing certain HTTP methods, such as POST, PUT, or CONNECT. Akl ¶120. Implementing such a rule, would have had the predictable benefit of achieving increased network

⁹ That these HTTP methods were known exfiltration methods is confirmed by the ’343 patent, which similarly recognized that it was well known that HTTP methods such as PUT and POST correspond to an exfiltration method. *See e.g.*, ’343 patent, 7:21-23, 7:55-60, 2:47-58.

Claim Language	Sourcefire
	security by protecting a network against known vulnerabilities. Akl ¶120. The Board in the IPR for the related '552 patent found that Sourcefire teaches creating this rule. Ex. 1042, 63 (agreeing that Sourcefire discloses that rules could be written that included the most common HTTP methods of GET, PUT, POST, and CONNECT as one or more of the rule criteria.). Patent Owner is therefore collaterally estopped here from relitigating this issue. As a second example, Sourcefire discloses using SSL/TLS rule keywords/arguments to invoke the application-layer SSL preprocessor and extract information on the SSL or TLS version from application packet headers.¹⁰ SSL and TLS are encryption protocols used to secure packets during communication across and/or between networks. • 827: “The ssl_version keyword can be used to match against version information for an encrypted session. <i>When a rule uses the ssl_version keyword, the rules engine invokes the SSL preprocessor to check traffic</i>

¹⁰ To the extent Patent Owner argues that **Sourcefire** obtains SSL and TSL version information from handshake fields, such an argument is irrelevant for the same reasons explained by the Board in the '552 patent IPR: the header of all post-handshake packets will have the same version value as identified in the handshake packet, and the Challenged Claims do not require that each packet in a session be inspected. Ex. 1042, 45-46. Patent Owner is collaterally estopped here from relitigating this issue.

Claim Language	Sourcefire														
	<i>for SSL version information.</i> For example, if you know there is a buffer overflow vulnerability in SSL version 2, you could use the ssl_version keyword with the sslv2 argument to identify traffic using that version of SSL.” 827-828: ssl_version Arguments <table> <tr> <th>Argument</th><th>Purpose</th></tr> <tr> <td>sslv2</td><td>Matches against traffic encoded using Secure Sockets Layer (SSL) Version 2.</td></tr> <tr> <td>sslv3</td><td>Matches against traffic encoded using Secure Sockets Layer (SSL) Version 3.</td></tr> </table> ssl_version Arguments (Continued) <table> <tr> <th>Argument</th><th>Purpose</th></tr> <tr> <td>tls1.0</td><td>Matches against traffic encoded using Transport Layer Security (TLS) Version 1.0.</td></tr> <tr> <td>tls1.1</td><td>Matches against traffic encoded using Transport Layer Security (TLS) Version 1.1.</td></tr> <tr> <td>tls1.2</td><td>Matches against traffic encoded using Transport Layer Security (TLS) Version 1.2.</td></tr> </table> 697: Adaptive profiles can also make the... SSL preprocessors more efficient by checking traffic for service identification.” 825: “Extracting SSL Information from a Session - You can use SSL rule keywords to invoke the Secure Sockets Layer (SSL) preprocessor and extract information about SSL version and session state from packets in an encrypted session. ... - The SSL preprocessor extracts state and version information from specific handshake fields. Two fields within the handshake indicate the version of SSL or TLS used to encrypt the session and the stage of the handshake.”	Argument	Purpose	sslv2	Matches against traffic encoded using Secure Sockets Layer (SSL) Version 2.	sslv3	Matches against traffic encoded using Secure Sockets Layer (SSL) Version 3.	Argument	Purpose	tls1.0	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.0.	tls1.1	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.1.	tls1.2	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.2.
Argument	Purpose														
sslv2	Matches against traffic encoded using Secure Sockets Layer (SSL) Version 2.														
sslv3	Matches against traffic encoded using Secure Sockets Layer (SSL) Version 3.														
Argument	Purpose														
tls1.0	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.0.														
tls1.1	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.1.														
tls1.2	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.2.														

Claim Language	Sourcefire
	• 826: “The ssl_state keyword can be used to match against state information for an encrypted session. When a rule uses the ssl_state keyword, the rules engine invokes the SSL preprocessor to check traffic for SSL state information.” • <i>See also</i> 491, 597-601. To the extent Sourcefire does not explicitly describe a rule in which the SSL or TLS version is specifically included as a keyword, a POSITA would have found obvious a rule using the ssl_version keyword to identify traffic using particular versions of SSL or TLS, and either pass or drop those packets. Sourcefire discloses that intrusion rules can be implemented by the SSL preprocessor and the rules engine and provides a specific keyword option to access the SSL and TLS version. Sourcefire, 827-828. For example, Sourcefire discloses that if there is a known vulnerability in SSL version 3, you could use the ssl_version keyword with the sslv2 argument to identify traffic using that version of SSL. <i>Id.</i> The security vulnerabilities of SSLv2 were widely known. Ex. 1016; Akl ¶120; <i>see also</i> Exs. 1037, 1039. Thus, a POSITA would have understood that certain SSL/TLS versions had security vulnerabilities (SSLv2, SSLv3, TLS 1.0) and that later TLS versions (TLS 1.1, 1.2) were more secure. These vulnerabilities in early versions of TLS and SSL exposed packets to potential exfiltration, as a malicious actor exploiting the vulnerabilities could circumvent the encryption and gain access to the data. In view of the teachings of Sourcefire, a POSITA would have been motivated and found it obvious to write an intrusion rule in which the ssl_version keyword is specifically included as a rule in order to detect and drop packets containing certain SSL/TLS versions. Akl ¶120. Implementing such a rule would have had the predictable benefit of achieving increased network security by protecting a network against known vulnerabilities. Akl ¶120.

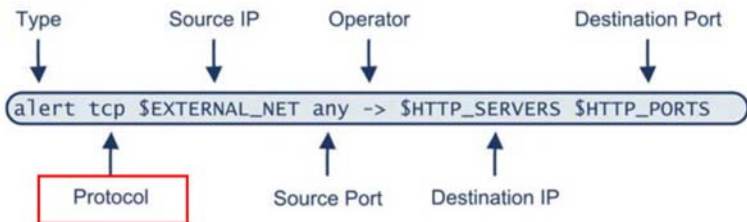
Claim Language	Sourcefire
	The Board in the IPR for the related '552 patent found that Sourcefire teaches creating such a rule. Ex. 1042, 56 (“We find that a person of ordinary skill would have been sufficiently motivated and informed by Sourcefire to write an intrusion rule with the ssl_version keyword to block packets whose SSL/TLS version matched the keyword....”). The Federal Circuit affirmed the Board’s decision, including its findings with respect to the using SSL/TLS rule keywords. Ex. 1043, 8 (“The SSL preprocessor extracts state and version information from specific handshake fields. Two fields within the handshake indicate the version of SSL or TLS used to encrypt the session and the stage of the handshake.”) (quoting J.A. 2252); <i>see also id.</i> (“Substantial evidence thus supports the Board’s finding that Sourcefire teaches the ‘determination’ limitations of the patent claims.”). Patent Owner is therefore collaterally estopped here from relitigating this issue. Implementation of the above-described rules involves examining application layer packet headers because SSL, TLS, and HTTP are application layer protocols. Akl ¶120; <i>see also</i> '343 patent, 2:40-47 (“an HTTP packet (e.g., an application-level HTTP packet) that specifies one or more specific HTTP methods (e.g., GET, PUT, POST, etc.)”), 8:15-16 (“the application packets contained in the IP packets may be TLS Record Protocol packets”). Moreover, rule keywords and arguments specifying certain HTTP methods and SSL versions require checking application layer packet headers because HTTP methods and SSL versions are specified in application header fields. For example, a POSITA would have recognized that HTTP methods are identified within the method field of an HTTP request header. Akl ¶120.
[1.e]/[8.g]/[15.e] responsive to determining the	Sourcefire discloses responsive to determining (e.g., “if the packet data matches all the conditions specified in a rule, the rule triggers”) the second portion of packets that comprises

Claim Language	Sourcefire
second portion of packets that comprises data corresponding to the second criterion specified by one or more operators specified by the at least one matching packet-filtering rule, [applying/ apply], [by the computing system and] to each packet in the second portion of packets [that match the second criterion], at least one packet transformation function configured to prevent an exfiltration operation, wherein the at least one packet transformation function indicates whether each	data corresponding to the second criterion specified by one or more operators specified by the at least one matching packet-filtering rule, applying, by the computing system and to each packet in the second portion of packets that match the second criterion (e.g., see [1.d]), at least one packet transformation function (e.g., “drop,” “pass”) configured to prevent an exfiltration operation (e.g., to stop certain “HTTP methods” such as “PUT,” “POST,” and “CONNECT” or use of certain unsecure versions of SSL/TLS), wherein the at least one packet transformation function indicates whether each packet in the second portion of packets is allowed to continue toward its destination (e.g., indicates “drop” or “pass”). The Board in the IPR for the related ’552 patent IPR found that Sourcefire teaches blocking packets that have application-header-field values corresponding to the criteria specified by the operator. Ex. 1042, 50-54. Patent Owner is therefore collaterally estopped here from relitigating this issue. Sourcefire discloses rules preventing exfiltration. For example, as discussed above with respect to [1.d], Sourcefire discloses rules including the most common HTTP methods of PUT, POST, and CONNECT as one or more of the rule criteria, and rules including SSL/TLS version as the rule criteria. Sourcefire, 568, 786. Also as discussed above with respect to [1.d], it was known that certain HTTP methods, such as PUT, POST, and CONNECT, and certain SSL/TLS versions, such as SSLv2, had known vulnerabilities and could be used for exfiltration, and it would have been obvious to

Claim Language	Sourcefire
packet in the second portion of packets is allowed to continue toward its destination.	create rules using the corresponding keywords to identify and drop such packets.¹¹ Sourcefire discloses a packet transformation function that indicates whether each packet that has met the first and second criteria (<i>i.e.</i>, a second portion of packets) is passed or dropped (<i>i.e.</i>, whether it is allowed to continue toward its destination): As discussed in [1.d], Sourcefire discloses that preprocessors and the rule engine compare application header field values of each packet against keywords and arguments specified in the rule arguments and that rule headers specify the rule’s action (<i>e.g.</i>, pass, drop). Therefore, responsive to a determination that an application header field value matches a second criterion in the rule, the 3D Sensor applies the rule’s action to each packet. • 435: “You can set a rule to Generate Events, Drop and Generate Events, or to Disable individually or you can filter the rules by a variety of factors to select the rules for which you want to modify the state. If you assign inline or inline with fail open interface sets to your IPS detection engines, you can <i>use the Drop and Generate Events rule state in inline intrusion deployments to drop malicious packets.</i>” • 761: “If it is a <i>pass rule</i>, it ignores the traffic. ... For a <i>drop rule</i> in an inline deployment, IPS drops the packet and generates an event.”

¹¹ This is consistent with the ’343 patent, which subsequently recognized that HTTP methods such as PUT and POST correspond to an exfiltration method. See *e.g.*, ’343 patent, 7:21-23, 7:55-60, 2:47-58.

Claim Language	Sourcefire
	765: “Specifying Rule Actions ... Each rule header includes a parameter that <i>specifies the action the system takes when a packet triggers a rule.</i> ... Rules with the action set to pass do not generate an event against, or log the details of, the packet that triggered the rule. ... In an inline deployment, rules with the rule state set to Drop and Generate Events generate an intrusion event against the packet that triggered the rule.” - See also 32, 560, 863. Akl ¶121.
[2.pre]/[9.pre]/[16.pre] The method of claim 1, wherein: The apparatus of claim 8, wherein The one or more non-transitory computer-readable media of claim 15, wherein:	See Claims 1/8/15 Akl ¶122.
[2.a]/[9.a]/[16.a] the first portion of packets comprises a packet header field value	Sourcefire discloses the first portion of packets (e.g., see [1.b]) comprises a packet header field value indicating a protocol type (e.g., “IP,” “TCP,” “UDP”) associated with a particular type of data transfer (e.g., “HTTP”), and corresponding to the first criterion specified by the at least one matching packet - filtering rule (e.g., see [1.b]).

Claim Language	Sourcefire
indicating a protocol type associated with a particular type of data transfer, and corresponding to the first criterion specified by the at least one matching packet - filtering rule.	Sourcefire discloses that the protocol type is indicated in a packet header field value. Sourcefire, 766. As discussed in [1.b], the field corresponds to a first criterion in the packet-filtering rule. A protocol type such as TCP is associated with a type of data transfer such as HTTP because HTTP packets are encapsulated in TCP/IP packets under the OSI model. Akl ¶123. 764:  766: <i>“In each rule header, you must specify the protocol of the traffic the rule inspects.</i> You can specify the following network protocols for analysis: - o ICMP (Internet Control Message Protocol) - o IP (Internet Protocol) - o TCP (Transmission Control Protocol) - o UDP (User Datagram Protocol) Use IP as the protocol type to examine all protocols assigned by IANA, including TCP, UDP, ICMP, IGMP, and many more.” Akl ¶123.
[3.pre]/[10.pre]/[17.pre] The method of claim 1, wherein:	See Claims 1/8/15 Akl ¶124.

Claim Language	Sourcefire
The apparatus of claim 8, wherein: The one or more non-transitory computer-readable media of claim 15, wherein:	
[3.a]/[10.a]/[17.a] the first portion of packets comprises data indicating a destination port number associated with a particular type of data transfer, and corresponding to the first criterion specified by the at least one matching packet filtering rule.	Sourcefire discloses the first portion of packets (<i>e.g.</i>, <i>see</i> [1.b]) comprises data indicating a destination port number (<i>e.g.</i>, “port number,” “destination ports”) associated with a particular type of data transfer (<i>e.g.</i>, “HTTP”), and corresponding to the first criterion specified by the at least one matching packet filtering rule (<i>e.g.</i>, <i>see</i> [1.b]). Sourcefire discloses that destination ports, which are fields in packet headers, are associated with data transfer protocols such as HTTP. Sourcefire further discloses that “destination ports are usually tied to a service” and that HTTP is such a service. Sourcefire, 1577, 427. This is consistent with the ’343 patent, which acknowledges that “HTTP and HTTPS” are examples of “data transfer protocols.” ’343, 8:59-65. Sourcefire additionally discloses rules that specify destination ports. Sourcefire, 764, 768. • 425: “You can filter the rules listed in the Rules page by several rule content items. For example, you can quickly retrieve a rule by searching for the rule SID. You can also <i>find all rules that inspect traffic going to a specific destination port.</i>” • 427:

Claim Language	Sourcefire												
	Rule Content Filters (Continued) <table><tr><th>To use this filter, click...</th><th>Then...</th><th>Result</th></tr><tr><td>Destination port</td><td>Type the destination port to filter by. The port value must be an integer between 1 and 65535 or a port variable.</td><td>Finds rules that include the specified destination port.</td></tr><tr><td>Rule Overhead</td><td>Select the amount of rule overhead to filter by.</td><td>Finds rules with the selected rule overhead.</td></tr><tr><td>Metadata</td><td>Type the metadata key-value pair to filter by, separated by a space. For example, type <code>metadata:"service http"</code> to locate rules with metadata relating to the HTTP service.</td><td>Find rules with metadata containing the matching key-value pair.</td></tr></table> 627: “The TCP protocol defines various states in which connections can exist. Each <i>TCP connection is identified by the source and destination IP addresses and source and destination ports</i>. TCP permits only one connection with the same connection parameter values to exist at a time.” 680: “Individual data types identify the sensitive data you can detect and generate events on in your specified destination network traffic. You can modify default settings for data type options that specify the following: ... - the <i>destination ports to monitor for each data type</i> 764: “Every standard text rule and shared object rule has parameters and arguments that make up the rule header. For example, the following illustrates the parts of a rule header: TypeSource IPOperatorDestination Port ↓ alert tcp \$EXTERNAL_NET any -> \$HTTP_SERVERS \$HTTP_PORTS ↑ ProtocolSource PortDestination IP	To use this filter, click...	Then...	Result	Destination port	Type the destination port to filter by. The port value must be an integer between 1 and 65535 or a port variable.	Finds rules that include the specified destination port.	Rule Overhead	Select the amount of rule overhead to filter by.	Finds rules with the selected rule overhead.	Metadata	Type the metadata key-value pair to filter by, separated by a space. For example, type <code>metadata:"service http"</code> to locate rules with metadata relating to the HTTP service.	Find rules with metadata containing the matching key-value pair.
To use this filter, click...	Then...	Result											
Destination port	Type the destination port to filter by. The port value must be an integer between 1 and 65535 or a port variable.	Finds rules that include the specified destination port.											
Rule Overhead	Select the amount of rule overhead to filter by.	Finds rules with the selected rule overhead.											
Metadata	Type the metadata key-value pair to filter by, separated by a space. For example, type <code>metadata:"service http"</code> to locate rules with metadata relating to the HTTP service.	Find rules with metadata containing the matching key-value pair.											

Claim Language	Sourcefire																																													
	Rule Header Values <table><thead><tr><th>Rule Header Component</th><th>Example Value</th><th>This Value...</th></tr></thead><tbody><tr><td>Action</td><td>alert</td><td>Generates an intrusion event when triggered.</td></tr><tr><td>Protocol</td><td>tcp</td><td>Tests TCP traffic only.</td></tr><tr><td>Source IP Address</td><td>\$EXTERNAL_NET</td><td>Tests traffic coming from any host that is not on your internal network.</td></tr><tr><td>Source Ports</td><td>any</td><td>Tests traffic coming from any port on the originating host.</td></tr><tr><td>Operator</td><td>-></td><td>Tests external traffic (destined for the web servers on your network).</td></tr><tr><td>Destination IP Address</td><td>\$HTTP_SERVERS</td><td>Tests traffic to be delivered to any host specified as a web server on your internal network.</td></tr><tr><td>Destination Ports</td><td>\$HTTP_PORTS</td><td>Tests traffic delivered to an HTTP port on your internal network.</td></tr></tbody></table> 768: “Rule headers specify the source and destination ports of the traffic the rule Inspects. You can <i>restrict packet inspection to packets originating from or destined to specific ports..</i>” Source/Destination Port Syntax <table><thead><tr><th>To Specify...</th><th>Use</th><th>Example</th></tr></thead><tbody><tr><td>any port</td><td>any</td><td>any</td></tr><tr><td>a specific port</td><td>the port number</td><td>80</td></tr><tr><td>a range of ports</td><td>a dash between the first and last port number in the range</td><td>80-443</td></tr><tr><td>all ports less than or equal to a specific port</td><td>a dash before the port number</td><td>-21</td></tr><tr><td>all ports greater than or equal to a specific port</td><td>a dash after the port number</td><td>80-</td></tr><tr><td>all ports except a specific port or range of ports</td><td>the ! character before the port, port list, or range of ports you want to negate Note that you can logically use negation with all port designations except <i>any</i>, which if negated would indicate <i>no port</i>.</td><td>!20</td></tr></tbody></table> 831: “Target ports are always <i>HTTP ports.</i>” 1577: “Because <i>destination ports are usually tied to a service</i>, this workflow can help you detect services that are experiencing an uncommonly high volume of alerts. The Destination Port column can also help you identify services that should not be present on your network.” 1608: “With IPS, if you are viewing intrusion events and want to constrain the list to only events where the <i>destination port is 80, click 80 (http)/tcp in the DST Port/ICMP Code column.</i>”	Rule Header Component	Example Value	This Value...	Action	alert	Generates an intrusion event when triggered.	Protocol	tcp	Tests TCP traffic only.	Source IP Address	\$EXTERNAL_NET	Tests traffic coming from any host that is not on your internal network.	Source Ports	any	Tests traffic coming from any port on the originating host.	Operator	->	Tests external traffic (destined for the web servers on your network).	Destination IP Address	\$HTTP_SERVERS	Tests traffic to be delivered to any host specified as a web server on your internal network.	Destination Ports	\$HTTP_PORTS	Tests traffic delivered to an HTTP port on your internal network.	To Specify...	Use	Example	any port	any	any	a specific port	the port number	80	a range of ports	a dash between the first and last port number in the range	80-443	all ports less than or equal to a specific port	a dash before the port number	-21	all ports greater than or equal to a specific port	a dash after the port number	80-	all ports except a specific port or range of ports	the ! character before the port, port list, or range of ports you want to negate Note that you can logically use negation with all port designations except <i>any</i> , which if negated would indicate <i>no port</i> .	!20
Rule Header Component	Example Value	This Value...																																												
Action	alert	Generates an intrusion event when triggered.																																												
Protocol	tcp	Tests TCP traffic only.																																												
Source IP Address	\$EXTERNAL_NET	Tests traffic coming from any host that is not on your internal network.																																												
Source Ports	any	Tests traffic coming from any port on the originating host.																																												
Operator	->	Tests external traffic (destined for the web servers on your network).																																												
Destination IP Address	\$HTTP_SERVERS	Tests traffic to be delivered to any host specified as a web server on your internal network.																																												
Destination Ports	\$HTTP_PORTS	Tests traffic delivered to an HTTP port on your internal network.																																												
To Specify...	Use	Example																																												
any port	any	any																																												
a specific port	the port number	80																																												
a range of ports	a dash between the first and last port number in the range	80-443																																												
all ports less than or equal to a specific port	a dash before the port number	-21																																												
all ports greater than or equal to a specific port	a dash after the port number	80-																																												
all ports except a specific port or range of ports	the ! character before the port, port list, or range of ports you want to negate Note that you can logically use negation with all port designations except <i>any</i> , which if negated would indicate <i>no port</i> .	!20																																												

Claim Language	Sourcefire
	• <i>See also</i> 401, 427, 465, 768-769, 870, 1222, 1275. Akl ¶125.
[4.pre]/[11.pre]/[18.pre] The method of claim 1, wherein The apparatus of claim 8, wherein The one or more non-transitory computer-readable media of claim 15, wherein	See Claims 1/8/15 Akl ¶126.
[4.a]/[11.a]/[18.a] the application header field value identifies one or more hypertext transfer protocol (HTTP) methods, wherein the second portion of packets comprises a plurality of HTTP packets, wherein the second portion of packets	Sourcefire discloses wherein the application header field value (<i>e.g.</i>, <i>see</i> [1.d]) identifies one or more hypertext transfer protocol (HTTP) methods (<i>e.g.</i>, “HTTP Methods”), wherein the second portion of packets (<i>e.g.</i>, <i>see</i> [1.d]) comprises a plurality of HTTP packets, wherein the second portion of packets comprises a third portion of packets and a fourth portion of packets, and wherein each packet of the third portion of packets comprises at least one GET method value (<i>e.g.</i>, “GET”), and each packet of the fourth portion of packets comprises at least one of a PUT method value (<i>e.g.</i>, “PUT”), a POST method value (<i>e.g.</i>, “POST”), or a CONNECT method value (<i>e.g.</i>, “CONNECT”). As discussed in [1.d], Sourcefire discloses invoking the HTTP Inspect preprocessor to identify HTTP methods such as GET, PUT, POST, and CONNECT in application header field values. A POSITA would have understood that in a plurality of HTTP method packets, one or more packets may variously

Claim Language	Sourcefire
comprises a third portion of packets and a fourth portion of packets, and wherein each packet of the third portion of packets comprises at least one GET method value, and each packet of the fourth portion of packets comprises at least one of a PUT method value, a POST method value, or a CONNECT method value, [the method further comprising:/and wherein the instructions, when executed by the at least one processor cause the apparatus to:/ and wherein the instructions, when executed	specify HTTP method options such as GET, PUT, POST, or CONNECT. Akl ¶127. Thus, the second portion of packets would comprise a third portion of packets (<i>i.e.</i>, packets having GET method values) and fourth portions of packets (<i>i.e.</i>, packets having PUT, POST, or CONNECT method values). • 560-561: “The <i>HTTP Inspect preprocessor</i> is responsible for: ○ <i>decoding and normalizing HTTP requests</i> sent to and HTTP responses received from web servers on your network ○ ... ○ making the normalized data available for additional rule processing • 568: “HTTP Methods Specifies <i>HTTP request methods for the preprocessor to inspect in addition to GET and POST.</i> Use a comma to separate multiple values.” • 786: “HTTP content Keyword Options Select this option... <i>HTTP Method</i> To <i>search for content matches in... the request method field, which identifies the action such as GET, PUT, CONNECT,</i> and so on to take on the resource identified in the URI.” • 807:

Claim Language	Sourcefire														
by the one or more computing devices, cause the one or more computing devices to:]	Snort-Specific Post Regular Expression Modifiers (Continued) <table> <tr> <th>Option</th><th>Description</th></tr> <tr> <td>H</td><td>Searches for the content within the header, excluding cookies, of an HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>D</td><td>Searches for the content within the header, excluding cookies, of a raw HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Raw Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>M</td><td>Searches for the content within the method field of a normalized HTTP request message decoded by the HTTP Inspect preprocessor; the method field identifies the action such as GET, PUT, CONNECT, and so on to take on the resource identified in the URI. See the content keyword HTTP Method option in the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>C</td><td>When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the normalized content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>K</td><td>When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the raw content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Raw Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.</td></tr> <tr> <td>S</td><td>Searches the 3-digit status code in an HTTP response. See the content keyword HTTP Status Code option in the HTTP content Keyword Options table on page 785 for more information.</td></tr> </table> Akl ¶127.	Option	Description	H	Searches for the content within the header, excluding cookies, of an HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.	D	Searches for the content within the header, excluding cookies, of a raw HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Raw Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.	M	Searches for the content within the method field of a normalized HTTP request message decoded by the HTTP Inspect preprocessor; the method field identifies the action such as GET, PUT, CONNECT, and so on to take on the resource identified in the URI. See the content keyword HTTP Method option in the HTTP content Keyword Options table on page 785 for more information.	C	When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the normalized content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.	K	When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the raw content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Raw Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.	S	Searches the 3-digit status code in an HTTP response. See the content keyword HTTP Status Code option in the HTTP content Keyword Options table on page 785 for more information.
Option	Description														
H	Searches for the content within the header, excluding cookies, of an HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.														
D	Searches for the content within the header, excluding cookies, of a raw HTTP request or response message decoded by the HTTP Inspect preprocessor. Note that you cannot use this option in combination with the content keyword HTTP Raw Header option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.														
M	Searches for the content within the method field of a normalized HTTP request message decoded by the HTTP Inspect preprocessor; the method field identifies the action such as GET, PUT, CONNECT, and so on to take on the resource identified in the URI. See the content keyword HTTP Method option in the HTTP content Keyword Options table on page 785 for more information.														
C	When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the normalized content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.														
K	When the HTTP Inspect preprocessor Inspect HTTP Cookies option is enabled, searches for the raw content within any cookie in an HTTP request header, and also within any set-cookie in an HTTP response header when the preprocessor Inspect HTTP Responses option is enabled. When Inspect HTTP Cookies is not enabled, searches the entire header, including the cookie or set-cookie data. Note that cookies included in the message body are treated as body content. Note also that you cannot use this option in combination with the content keyword HTTP Raw Cookie option to search the same content. See the HTTP content Keyword Options table on page 785 for more information.														
S	Searches the 3-digit status code in an HTTP response. See the content keyword HTTP Status Code option in the HTTP content Keyword Options table on page 785 for more information.														
[4.b]/[11.b]/[18.b] [allowing/allow] each packet of the third portion of packets to continue toward	Sourcefire discloses allowing (e.g., “pass”) each packet of the third portion of packets (e.g., packets indicating “GET”) to continue toward its respective destination (e.g., “pass”). As discussed in [1.d], [1.e], and [4.a], Sourcefire discloses invoking the HTTP Inspect preprocessor to identify HTTP methods such as GET, PUT, POST, and CONNECT in application header field values, and passing or dropping														

Claim Language	Sourcefire
its respective destination; and	(depending on how the rule is configured) packets that match the specified criteria. At a minimum, it would have been obvious in view of these disclosures to write a rule using the HTTP Method option of the HTTP content keyword as part of the application-layer rule criteria to invoke the HTTP inspect preprocessor to identify a packet using the “GET” HTTP method (third portion of packets) and to allow such a packet to continue towards its destination. A POSITA would have been motivated to write such a rule because it was well-known that the HTTP GET option is typically not exfiltration related. Akl ¶128; Ex. 1019, 13:65-67. Moreover, as discussed in [1.e], writing such a rule merely involves selecting from a finite number of criteria that are relevant to a selected network vulnerability, and would yield predictable results of increased network security. Additionally, implementing the rule would involve nothing more than applying a known technique to a known device. Akl ¶128.
[4.c]/[11.c]/[18.c] [blocking/block] each packet of the fourth portion of packets from continuing toward its respective destination.	Sourcefire discloses blocking (e.g., “drop”) each packet of the fourth portion of packets from continuing toward its respective destination. Similarly, at a minimum, it would have been obvious in view of these disclosures to write a rule using the HTTP Method option of the HTTP content keyword as part of the application-layer rule criteria to invoke the HTTP inspect preprocessor to identify a packet using the “PUT,” “POST,” or “CONNECT” HTTP methods (fourth portion of packets) and to block such a packet from continuing towards its destination. Akl ¶129. A POSITA would have been motivated to write such a rule because it was well known that PUT, POST, and CONNECT methods transmit data outside the network, as discussed in [1.d]. Sourcefire, 934; Akl ¶129. Moreover, as discussed in [1.e], writing such a rule merely involves selecting from a finite number of criteria that are

Claim Language	Sourcefire
	relevant to a selected network vulnerability, and would yield predictable results of increased network security. Additionally, implementing the rule would involve nothing more than applying a known technique to a known device. Akl ¶129.
[5.pre]/[12.pre] The method of claim 1, the method further comprising: The apparatus of claim 8, wherein the instructions, when executed by the one or more computing devices, cause the apparatus to:	See Claims 1/8 Akl ¶130.
[5.a]/[12.a] [allowing/allow], based on a determination that the application header field value indicates a request for [specified resource data/data from a specified resource], each packet of the second portion of packets	Sourcefire discloses allowing (<i>e.g.</i>, “pass”), based on a determination that the application header field value (<i>e.g.</i>, <i>see</i> [1.d]) indicates a request for specified resource data (<i>e.g.</i>, “GET”), each packet of the second portion of packets indicating a request for specified resource data (<i>e.g.</i>, packets indicating “GET”) to continue toward its respective destination (<i>e.g.</i>, “pass”). <i>See</i> [4.a]/[11.a] and [4.b]/[11.b]. Akl ¶131.

Claim Language	Sourcefire
indicating a request for [specified resource data/data from a specified resource] to continue toward its respective destination; and	
[5.b]/[12.b] [blocking/block], based on a determination that the application header field value indicates a request to submit data to be processed by a specified resource, each packet of the second portion of packets indicating a request to submit data to be processed by [the/a] specified [resource] from continuing toward its respective destination.	Sourcefire discloses blocking (<i>e.g.</i>, “drop”), based on a determination that the application header field value (<i>e.g.</i>, <i>see</i> [1.d]) indicates a request to submit data to be processed by a specified resource (<i>e.g.</i>, “PUT,” POST,” “CONNECT”), each packet of the second portion of packets indicating a request to submit data to be processed by the specified resource (<i>e.g.</i>, packets indicating “PUT,” “POST,” “CONNECT”) from continuing toward its respective destination (<i>e.g.</i>, “drop”). <i>See</i> [4.a]/[11.a] and [4.c]/[11.c]. Akl ¶132.

Claim Language	Sourcefire
[6.pre]/[13.pre]/[19.pre] The method of claim 1, further comprising The apparatus of claim 8, wherein the instructions, when executed by the one or more computing devices, cause the apparatus to The one or more non-transitory computer-readable media of claim 15, further comprising	See Claims 1/8/15 Akl ¶133.
[6.a]/[13.a]/[19.a] [dropping/drop] packets associated with a particular type of data transfer responsive to a determination that the packets associated with the particular type of data transfer	Sourcefire discloses dropping packets (<i>e.g.</i>, “drop”) associated with a particular type of data transfer (<i>e.g.</i>, “SSL”) responsive to a determination (<i>e.g.</i>, “if the packet data matches all the conditions specified in a rule, the rule triggers”) that the packets associated with the particular type of data transfer comprise data corresponding to a particular transport layer security (TLS) version value (<i>e.g.</i>, “TLS Version 1.0,” “TLS Version 1.1,” “TLS Version 1.2”). SSL is a type of data transfer. Akl ¶134. As discussed above with respect to [1.d] and [1.e], Sourcefire discloses a rule using an SSL keyword to identify SSL or TLS versions in

Claim Language	Sourcefire														
comprise data corresponding to a particular transport layer security (TLS) version value	packets using the SSL preprocessor to drop or allow packets, and passing or dropping (depending on how the rule is configured) packets that match the specified criteria. 827-828: “The ssl_version keyword can be used to match against version information for an encrypted session. When a rule uses the ssl_version keyword, the rules engine invokes the SSL preprocessor to check traffic for SSL version information. For example, if you know there is a buffer overflow vulnerability in SSL version 2, you could use the ssl_version keyword with the sslv2 argument to identify traffic using that version of SSL. Use a comma-separated list to specify multiple arguments for the SSL version. When you list multiple arguments, IPS evaluates them using the OR operator. For example, if you wanted to identify any encrypted traffic that was not using SSLv2, you could add ssl_version:ssl_v3, tls1.0, tls1.1, tls1.2 to a rule. The rule would evaluate any traffic using SSL Version 3, TLS Version 1.0, TLS Version 1.1, or TLS Version 1.2.” ssl_version Arguments (Continued) <table> <tr> <th>Argument</th><th>Purpose</th></tr> <tr> <td>tls1.0</td><td>Matches against traffic encoded using Transport Layer Security (TLS) Version 1.0.</td></tr> <tr> <td>tls1.1</td><td>Matches against traffic encoded using Transport Layer Security (TLS) Version 1.1.</td></tr> <tr> <td>tls1.2</td><td>Matches against traffic encoded using Transport Layer Security (TLS) Version 1.2.</td></tr> </table> ssl_version Arguments <table> <tr> <th>Argument</th><th>Purpose</th></tr> <tr> <td>sslv2</td><td>Matches against traffic encoded using Secure Sockets Layer (SSL) Version 2.</td></tr> <tr> <td>sslv3</td><td>Matches against traffic encoded using Secure Sockets Layer (SSL) Version 3.</td></tr> </table> As discussed in [1.d], at a minimum, it would have been obvious in view of Sourcefire to write a rule using the ssl_version keyword as part of the application-layer rule	Argument	Purpose	tls1.0	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.0.	tls1.1	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.1.	tls1.2	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.2.	Argument	Purpose	sslv2	Matches against traffic encoded using Secure Sockets Layer (SSL) Version 2.	sslv3	Matches against traffic encoded using Secure Sockets Layer (SSL) Version 3.
Argument	Purpose														
tls1.0	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.0.														
tls1.1	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.1.														
tls1.2	Matches against traffic encoded using Transport Layer Security (TLS) Version 1.2.														
Argument	Purpose														
sslv2	Matches against traffic encoded using Secure Sockets Layer (SSL) Version 2.														
sslv3	Matches against traffic encoded using Secure Sockets Layer (SSL) Version 3.														

Claim Language	Sourcefire
	criteria to invoke the SSL preprocessor to identify a packet using particular transport layer security (TLS) version and to drop such a packet. Akl ¶134.

B. Ground 2: Claims 7, 14, and 20 Are Rendered Obvious Over Sourcefire in View of Mantripragada

Sourcefire renders obvious claims 1, 8, and 15, as discussed in §VIII.A, above. Claims 7, 14, and 20 depend from each of those claims, respectively, and further recite “[dropping/drop] packets associated with a particular type of data transfer responsive to a determination that the packets associated with the particular type of data transfer comprise data corresponding to a particular real-time transport protocol value.” **Mantripragada** discloses this element. Akl ¶¶135-138.

1. Overview of Mantripragada

U.S. Pat. App. Publication 2009/0103524 (“**Mantripragada**”) is directed to enforcing security rules for Voice-over-IP (VoIP) and Unified Communications (UC) technology in IP communication networks. **Mantripragada**, Abstract, ¶¶2-3. For example, **Mantripragada** teaches monitoring Real-Time Transport Protocol (RTP) streams to ensure that various RTP values are not tampered with while the RTP session is active. *Id.* ¶¶69-70. RTP values that are checked include RTP protocol header sequence number (16 bits), RTP protocol header timestamp (32

bits), RTP protocol header synchronization source identification (32 bits), version (2 bits), padding (1 bit), extension (1 bit), CSRC (Contributing Source) count (4 bits), marker (1 bit), and payload type (7 bits). *Id.* ¶70. A POSITA would have understood that each of these variables is located within the header of an RTP packet. Akl ¶137; Ex. 1012, 24:55-67. **Mantripragada** further teaches rejecting (*i.e.*, dropping) packets that have RTP values that are out of compliance. *Id.* ¶70 (“Media streams that are not compliant with existing session flows ... are rejected....”).

Thus, **Mantripragada** discloses dropping packets (*e.g.*, “reject[ing]”) associated with a particular type of data transfer (*e.g.*, associated with “RTP ... streams”) responsive to a determination (*e.g.*, “are not compliant with existing session flows”) that the packets associated with the particular type of data transfer comprise data corresponding to a particular real-time transport protocol value (*e.g.*, corresponding to RTP values, including “version” and Contributing Source,” that are “out of compliance”).

Akl ¶¶111, 135-138.

2. Motivation to Combine

Sourcefire and **Mantripragada** are in the same field of art and are analogous art to the '343 patent—all are in the same fields of packet filtering and

network security. *E.g.*, Sourcefire, 33-35; Mantripragada, Abstract, ¶¶2, 69-70; '343, Abstract, 1:18-51; Akl ¶139.

A POSITA would have been motivated to apply **Mantripragada**'s teachings of dropping packets based on non-compliant RTP values in implementing an intrusion rule of **Sourcefire** in order to prevent attacks on RTP streams. Akl ¶140. For example, **Mantripragada** discloses that “[a]ttacks against RTP are applicable to virtually any VoIP or SIP environment” and identifies “Man-in-the-Middle (MITM) attacks” as an example. Mantripragada ¶69. The solution taught by **Mantripragada**—dropping packets based on out of compliant RTP values—prevents those attacks from succeeding by “ensur[ing] that [the packets] are not tampered while the session is active.” *Id.*; Akl ¶140.

A POSITA would have had a reasonable expectation of success in using **Mantripragada**'s teachings of dropping packets based on non-compliant RTP values in implementing an intrusion rule of **Sourcefire**, and would have known that such a combination (yielding the claimed limitations) would predictably work and provide the expected functionality. Akl ¶141. For example, **Sourcefire** discloses decoding UDP traffic (Sourcefire, 258)—the transport layer that RTP sits atop—and discloses application layer decoders and preprocessors (*id.*, 256), which would be used to decode RTP packets. Indeed, the 3D Sensor is able to detect “Real-time Transport Protocol.” *Id.*, 2034. The device will also “identify a session”

of a UDP stream (*id.*, 645-46), and the “flow” keyword is used to “select packets for inspection by a rule based on session characteristics” (*id.*, 820). **Sourcefire** further discloses that “you can create your own custom standard text rules.” *Id.*, 861. To the extent a specific rule keyword is not pre-configured to target the RTP values identified by **Mantripragada**, **Sourcefire** discloses a “byte_extract keyword” that reads a specified number of bytes from a packet with a specified offset (*id.*, 840-41), which a POSITA would have known to use in conjunction with RTP’s standardized, fixed-size header values (Ex. 1012, 24:55-67) to reliably extract the required values. Akl ¶141.

IX. SECONDARY CONSIDERATIONS

Petitioner is unaware of evidence of secondary considerations relevant to the Challenged Claims at the date of this filing. Akl ¶¶142-43. While Patent Owner raised evidence of alleged secondary considerations in IPRs challenging the related ’552 and ’193 patents, the Board rejected Patent Owner’s evidence. Ex. 1042, 56-62. The Federal Circuit affirmed, holding that “the objective indicia of nonobviousness were not entitled to substantial weight.” Ex. 1043, 9.

X. CONCLUSION

Substantial, new, and noncumulative technical teachings have been presented for each Challenged Claim, which are disclosed and/or rendered obvious for the reasons set forth above. Akl ¶¶112-115, 144-148. There is a reasonable

likelihood that Petitioner will prevail as to each of these Challenged Claims. *Inter partes* review of Claims 1-20 of the '343 patent is accordingly requested.

Respectfully submitted,

Dated: July 6, 2021

By: /Scott A. McKeown/
Scott A. McKeown
Registration No. 42,866

Counsel for Petitioner Palo Alto Networks,
Inc.

CERTIFICATE OF COMPLIANCE

Pursuant to 37 C.F.R. § 42.24(a) and (d), the undersigned hereby certify that the Petition For *Inter Partes* Review complies with the type-volume limitation of 37 C.F.R. § 42.24(a)(i) because, exclusive of the exempted portions, it contains 13,913 words as counted by the word processing program used to prepare the paper.

Dated: July 6, 2021

By: /Christopher M. Bonny/
ROPES & GRAY LLP

CERTIFICATE OF SERVICE

The undersigned certifies service pursuant to 37 C.F.R. §§ 42.6(e) and 42.105(b) on the Patent Owner by FedEx of a copy of this Petition for *Inter Partes* Review and supporting materials at the correspondence address of record for the '343 patent:

BANNER & WITCOFF, LTD.
1100 13th STREET, N.W.
SUITE 1200
WASHINGTON DC 20005-4051

Dated: July 6, 2021

By: /Jonathan Bradford/
Name: Jonathan Bradford
ROPES & GRAY LLP