

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

TREND MICRO, INC.
Petitioner,

v.

WEBROOT, INC.
Patent Owner.

IPR2023-01392

U.S. Patent No. 8,181,244

DECLARATION OF SETH JAMES NIELSON

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	QUALIFICATIONS	1
III.	MATERIALS CONSIDERED	12
IV.	LEGAL STANDARDS	14
V.	Technical Background that Would Have Been Known to A POSITA	16
A.	Log aggregation	17
B.	System calls, hooking, and the Windows API.....	18
C.	Windows monitoring, kernel drivers, and APIs.....	21
D.	Behavior monitoring techniques for anti-malware	23

E.	Malware and Privilege Escalation	29
VI.	OVERVIEW OF THE CHALLENGED PATENT	30
A.	THE ‘244 PATENT SPECIFICATION AND CLAIMS.....	30
B.	PROSECUTION HISTORY.....	32
VII.	LEVEL OF SKILL IN THE ART	35
VIII.	Overview of the prior art.....	35
A.	OVERVIEW OF CHEN	35
B.	OVERVIEW OF PICCARD.....	37
C.	OVERVIEW OF LI	40
IX.	GROUND 1: Claims 1-16 are rendered obvious by LI in view of Piccard	44
A.	Motivation to Combine Li and Piccard.....	44
B.	Claim 1 is rendered obvious by Li in view of Piccard.....	48
1.	Limitation 1[a]	48
2.	Limitation 1[b]	50
3.	Limitation 1[c]	52
4.	Limitation 1[d]	54
5.	Limitation 1[e]	55
6.	Limitation 1[f]	57
7.	Limitation 1[g]	58
8.	Limitation 1[h]	61
C.	Claim 2 is rendered obvious by Li in view of Piccard.....	63
1.	Limitation 2[a]	63
2.	Limitation 2[b]	64
3.	Limitation 2[c]	65
D.	Claim 3 is rendered obvious by Li in view of Piccard.....	66

E.	Claim 4 is rendered obvious by Li in view of Piccard.....	67
F.	Claim 5 is rendered obvious by Li in view of Piccard.....	68
G.	Claim 6 is rendered obvious by Li in view of Piccard.....	69
H.	Claim 7 is rendered obvious by Li in view of Piccard.....	72
I.	Claim 8 is rendered obvious by Li in view of Piccard.....	73
J.	Claim 9 is rendered obvious by Li in view of Piccard.....	74
	1. Limitation 9[a]	74
	2. Limitation 9[b]	76
	3. Limitation 9[c]	76
	4. Limitation 9[d]	79
	5. Limitation 9[e]	80
	6. Limitation 9[f]	81
	7. Limitation 9[g]	81
K.	Claim 10 is rendered obvious by Li in view of Piccard.....	83
L.	Claim 11 is rendered obvious by Li in view of Piccard.....	84
M.	Claim 12 is rendered obvious by Li in view of Piccard.....	86
N.	Claim 13 is rendered obvious by Li in view of Piccard.....	88
O.	Claim 14 is rendered obvious by Li in view of Piccard.....	89
P.	Claim 15 is rendered obvious by Li in view of Piccard.....	89
Q.	Claim 16 is rendered obvious by Li in view of Piccard.....	89
X.	GROUND 2: Claims 1-16 are rendered obvious by CHEN in view of Piccard	90
	A. Motivation to Combine Chen and Piccard.....	90
	B. Claim 1 is rendered obvious by Chen in view of Piccard.....	93
	1. Limitation 1[a]	93
	2. Limitation 1[b]	96

3.	Limitation 1[c]	98
4.	Limitation 1[d]	100
5.	Limitation 1[e]	103
6.	Limitation 1[f]	104
7.	Limitation 1[g]	106
8.	Limitation 1[h]	109
C.	Claim 2 is rendered obvious by Chen in view of Piccard	111
1.	Limitation 2[a]	111
2.	Limitation 2[b]	112
3.	Limitation 2[c]	115
D.	Claim 3 is rendered obvious by Chen in view of Piccard	116
E.	Claim 4 is rendered obvious by Chen in view of Piccard	117
F.	Claim 5 is rendered obvious by Chen in view of Piccard	118
G.	Claim 6 is rendered obvious by Chen in view of Piccard	119
H.	Claim 7 is rendered obvious by Chen in view of Piccard	120
I.	Claim 8 is rendered obvious by Chen in view of Piccard	122
J.	Claim 9 is rendered obvious by Chen in view of Piccard	123
1.	Limitation 9[a]	123
2.	Limitation 9[b]	126
3.	Limitation 9[c]	126
4.	Limitation 9[d]	127
5.	Limitation 9[e]	129
6.	Limitation 9[f]	130
7.	Limitation 9[g]	130
K.	Claim 10 is rendered obvious by Chen in view of Piccard	132

L.	Claim 11 is rendered obvious by Chen in view of Piccard.....	132
M.	Claim 12 is rendered obvious by Chen in view of Piccard.....	133
N.	Claim 13 is rendered obvious by Chen in view of Piccard.....	134
O.	Claim 14 is rendered obvious by Chen in view of Piccard.....	135
P.	Claim 15 is rendered obvious by Chen in view of Piccard.....	136
Q.	Claim 16 is rendered obvious by Chen in view of Piccard.....	136
XI.	There are no objective indicia of nonobviousness.....	136
XII.	Conclusion	137

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

I. INTRODUCTION

2. I have been retained by counsel for Trend Micro, Inc. (“Trend Micro” or “Petitioner”) as a technical expert in connection with the proceeding identified above. I submit this Declaration in support of Petitioner Trend Micro’s Petition for Inter Partes Review (the “Petition”) of United States Patent No. 8,181,244 (“the ‘244 patent”) against Webroot, Inc. (“Webroot” or “Patent Owner”). All “Ex. 10XX” cites herein are to the Exhibits to the Petitions. All “Ex. _” cites followed by a letter are to exhibits to this Declaration. All citations to “Section XX” are internal citations to the sections of this Declaration. All citations to “Paragraph XX” are internal citations to the paragraphs of this Declaration.

3. I have been asked to consider whether the claims of the ‘244 patent are obvious in light of the prior art discussed below. The ‘244 patent is directed to general application of known machine learning techniques to the identification of malware.

4. In summary, my conclusion is that the claims of the ‘244 patent are obvious in light of the prior art discussed below. As set forth in detail in this declaration, I have reached the following conclusions:

- a. Claims 1-16 are rendered obvious by the combination of Li and Piccard.
- b. Claims 1-16 are rendered obvious by the combination of Chen and Piccard.

II. QUALIFICATIONS

5. I am a subject matter expert in cybersecurity, computer networks, and software engineering. I am the Founder and Chief Scientist of Crimson Vista, a computer security research and engineering company. Furthermore, I hold appointments at the University of Texas at Austin

as an Adjunct Associate Professor in the department of Computer Science and as a Cybersecurity Fellow in the Robert Strauss Center for International Security and Law.

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

7. A copy of my current C.V. is attached hereto as **Exhibit 1004**. My C.V. includes all of my relevant education and work experience, a list of all publications that I have authored in the previous ten years, as well as a list of all cases in which, during the previous four years, I have testified as an expert at trial or by deposition. My experience most relevant to the opinions offered in this report is summarized below.

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

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

space that could be farmed out to the worker nodes for processing. I also studied artificial intelligence and machine learning.

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

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

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

13. While working at Lineo/Metrowerks, I also returned to BYU to pursue my Master's degree in Computer Science. In addition to the graduate level course work in wireless computer networks and compilers, I pursued graduate research in software engineering topics, with a special

emphasis on how programmers think while creating and modifying code. During my course work, I took a special topics class called “Programmer Cognition” as well as a graduate-level neuroscience class from the Psychology department.

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

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

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

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

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

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

20. My Ph.D. Thesis was entitled "Designing Incentives for Peer-to-Peer Systems," and it built on this concept. Given a network where participants cannot be forced to cooperate, the operation of said network must induce cooperation by design of the outcomes. In other words, it must be in each participant's best interest to contribute to the cooperative operation. I conducted experiments including simulated extensions to the BitTorrent peer-to-peer protocol for long-term identities and mechanisms for cooperative anonymity. These systems were predecessors to modern cryptocurrency and distributed ledger technologies.

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

certification. The encryption library provided advanced operations such as secure data splitting and recovery.

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

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

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

25. In 2011, I began work as a Research Scientist at Harbor Labs and continued with that consulting firm until fall 2015. I worked with a wide range of clients, specializing in network security, network communications, software architecture, and programming languages. I analyzed an extensive collection of commercial software, including software related to secure email, cloud-based multimedia delivery, document signing, anti-virus and anti-intrusion, high-performance routing, networking protocol stacks in mobile devices, PBX telecommunications software, VoIP, and peer-to-peer communications. I also analyzed security considerations for potential technology acquisitions.

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

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

28. For other clients, I have “resurrected” or re-created legacy software systems. For example, I helped one client make remote desktop-sharing/viewing systems from the mid 90’s operational. I helped them identify the most compatible components from an old CVS repository, obtain the necessary legacy hardware and software to rebuild the source code, and diagnose why the separate components weren’t completely compatible with each other. Using tools from the era (i.e., the mid-90’s), I identified and fixed these issues in C++ and Java code, and successfully demonstrated the operational system across a small cluster of networked virtual machines.

29. During my final year at Harbor Labs, I was engaged as the principal consultant with a large biomedical device firm in a twelve-month analysis of the security of their products. Notably, medical devices were for some time not considered significant threats in terms of computer security. However, recent demonstrations by security researchers of the various ways in which a malicious individual might harm a person using a medical device has shifted the thinking in the industry. Accordingly, I was engaged to assist this company in the analysis of their products,

their process, and their future roadmap in order to ensure that patients are not harmed. I and my team analyzed design documents, hardware, and a broad range of additional resources in order to expose potential problems. The security of these systems depends, in part, on the architecture and deployment of the networks in which they operate.

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

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

32. For example, I have been retained by a start-up in telecommunications security to provide cryptography expertise and evaluations of their protocols and architectures. My team and I have prototyped new protocols, written up analyses, and presented to potential partners and investors.

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

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

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

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

37. Through Crimson Vista, I also invest in research and development. Recent projects include engaging with a partner to implement prototypes of communications security protocols for next-generation automobiles. I also gave a talk on “Detecting Malicious Sandboxes” at the Workshop on Defensive Deception and Trust in Autonomy, in association with the 2018 Naval Applications of Machine Learning Workshop.

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

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

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

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

42. At Johns Hopkins University I taught Network security and Advanced Network Security. I created a custom curriculum and lab experience wherein students developed their own security protocols as a class and then attempted to break their own creations. Students learned how hard it is to get security right, and how easy it is to find something wrong. I published a paper about the labwork in the Journal of Computer Science Education entitled, “PLAYGROUND: Preparing Students for the Cyber Battleground.”

43. Beyond course instruction, I also mentored Masters students at Johns Hopkins in their capstone projects. These projects included networking security and privacy concerns across a wide range of technologies including cryptography, drone security, iOS security, BitCoin, SSL

vulnerabilities, and Twitter “botnets.” My students and I have published multiple papers from these capstone projects.

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

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

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

47. I am also the co-founder and current director of the Crypto Done Right project. This project is currently hosted by Johns Hopkins University and funded by a grant from Cisco. I am transitioning the project to be hosted by my company Crimson Vista or spun out into a completely independent non-profit entity. Crypto Done Right is designed to bridge the gap between cryptography SME’s and the IT professionals that use it. It provides authoritative guidance on deployment, lifecycle, and management of cryptography in IT systems, software development, and technical management.

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

III. MATERIALS CONSIDERED

49. I have considered information from various sources in forming my opinions. Besides drawing from approximately two decades of experience in the computer industry, I also have reviewed the ‘244 patent and its file history, the other documents and references as cited herein, and general technical references. I understand that Counsel for Petitioner have submitted a Petition that reference by exhibit number. For simplicity and convenience, I have adopted the same exhibit numbering.

50. The exhibits are:

- Ex. 1001: U.S. Patent No. 8,181,244 to Boney. (“the ‘244 patent”)
- Ex. 1002: Prosecution history of the ‘244 patent
- Ex. 1003: Expert Declaration of Dr. Seth James Nielson
- Ex. 1004: Curriculum Vitae of Dr. Seth James Nielson
- Ex. 1005: Webroot, Inc. and Open Text, Inc. v. Trend Micro Inc., Case No. 6:22-cv-00243-ADA-DTG, Claim Construction Order, Dkt. 391 (W.D. Tex., July 25, 2023).
- Ex 1006: Kevin Chen, ECF – Event Correlation for Forensics, Australian Computer, Network & Information Forensics Conference (2003).
- Ex. 1007: U.S. Patent Publication No. 2007/0006310 to Piccard. (“Piccard”)
- Ex. 1008: Zhenmin Li, UCLog: A Unified, Correlated Logging Architecture for Intrusion Detection, Int’l Conf. on Telecomm. Sys.- Modelling & Analysis (2004).

- Ex. 1009: Declaration of Ingrid Hsieh-Yee, Ph.D.
- Ex. 1010: Peter G. Viscarola, Windows NT Device Driver Development, ORS Open Systems Resources, Inc., (1999).
- Ex. 1011: Nicholas Weaver, Worms vs. Perimeters: The Case or Hard-LANS, 12th Annual IEEE Symposium (Aug. 2004).
- Ex. 1012: Weidong Cui, Design and Implementation of an Extrusion-based Break-In Detector for Personal Computers (2006).
- Ex. 1013: Cristina Abad, Log Correlation for Intrusion Detection: A Proof of Concept, 19th Annual Computer Security Applications Conference (2003).
- Ex. 1014: Robert Rinnan, Benefits of Centralized Log File Correlation, Gjøvik University College (2005).
- Ex. 1015: Ed Skoudis, Malware: Fighting Malicious Code, Pearson (Nov. 2003).
- Ex. 1016: Platform SDK: Windows Management Instrumentation, Win32_Process, Win32_Process class [WMI] (archive.org) (Jul. 17, 2004).
- Ex. 1017: Matthew M. Lavy, Windows Management Instrumentation (WMI), Sams (Oct. 2001).
- Ex. 1018: Yariv Kaplan, API Spying Techniques for Windows 9x, NT and 2000, API Spying Techniques for Windows 9x, NT and 2000 (archive.org) (Feb. 10, 2003).
- Ex. 1019: Stephanie Forrest, Computer Immunology, Communications of the ACM 40(1), (Oct. 1997).

IV. LEGAL STANDARDS

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

52. I have been informed that in a parallel district court litigation the owner of the '244 patent is arguing that the '244 patent claims should be afforded their plain and ordinary meaning. I have also been requested to adopt patent owner's position on claim construction for the purposes of rendering my opinions set forth in this declaration.

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

54. I have been informed and understand that the subject matter of a patent claim is obvious if the differences between the subject matter of the claim and the prior art are such that the subject matter of the claim as a whole would have been obvious at the effective filing date to a person having ordinary skill in the art to which the subject matter pertains. I have also been informed that the framework for determining obviousness involves considering the following factors: (i) the scope and content of the prior art; (ii) the differences between the prior art and the claimed subject matter; (iii) the level of ordinary skill in the art; and (iv) any objective evidence of non-obviousness.

55. I further understand that the claimed subject matter would have been obvious to one of ordinary skill in the art if, for example, it results from the combination of known elements according to known methods to yield predictable results, the simple substitution of one known element for another to obtain predictable results, the use of a known technique to improve similar

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

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

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

design incentives or market forces if the variations would have been predictable to a POSITA. I also understand that a need or problem known in the field of endeavor can provide a reason for combining elements in the manner claimed.

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

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

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

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

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

62. In this section of my Declaration, I discuss some of the background and context for the alleged invention in the ‘244 patent that would have been known to a POSITA as of the filing date of the ‘244 patent, which I understand to be April 20, 2006. Any discussion of the background technologies in this section is written to be as understood by a POSITA of that date.

A. Log aggregation

63. It was well understood by practitioners of the art at the time of the alleged invention that correlating logs from disparate sources improved intrusion detection.

The problem faced today by system administrators are the correlation of Firewall logs, intrusion detection system event logs, operating system event logs, mail system logs, database logs, web server logs, antivirus logs, router/switch logs etc. [10] states that potentially, all these logs can identify a threat, and may receive hundreds or thousands of entries a day. The examination of these logs is often performed by staff short of time and knowledge, and for a company; resources may be a limitation. ***Despite the resource limitations we want log file correlation because it improves intrusion detection*** [12].
(Ex. 1014, p. 15, emphasis added).

64. Many commercial products were available that had this capability, as well as related capabilities necessary to make such correlation function well, such as aggregation:

Reading some of the whitepapers and briefings on products web-pages provides some insight into their architecture, and motivation. There seem to be differences among the products, but ***they all seem to follow a layout consisting of normalization, aggregation, correlation, and visualization***. As an example the netForensics product [17] automatically aggregates events from thousands of disparate sources into a single data format, normalizes the events into 100 event types across nine categories, making it possible to automatically identify events using correlation, generate reports, and perform analysis.
(*Id.*, p. 15, emphasis added).

65. Rinnan gives an overview of “the approaches found in the literature”, several of which are centralization, consolidation, and aggregation. (*Id.*, pp. 16-17).

The centralization process may be defined as gathering the log files/entries in one server.
(*Id.*, p. 16).

Consolidation is the process that takes data from different systems and entities, and possibly disparate formats, and combines and aggregates that information to create a unified view [42].
(*Id.*, pp. 16-17).

Data aggregation [2, 23] organizes normalized data by category, for instance IT systems, applications, etc. [4] states that it is necessary to

eliminate redundant or duplicate event data from the security event data stream. This includes refining and optimizing the amount of information that is presented to security analysts. ... Grouping similar events together and providing answers to how many times an attack happened over a certain time period, enables security experts to group and count these events quickly by port, source IP, target IP, business asset (e.g. Web servers or mail servers.) or other parameters [11]. [19] provides filtering examples such as protocol type, time, IP, and MAC Address.
(*Id.*, p. 17).

66. Thus, a POSITA would have understood the value of collecting various types of log information to a centralized location and aggregating it. A POSITA also would have been aware that this common technique was commonly being explored, developed, and improved. For example, Chen, one of the prior art references, looked to fill a gap over software such as netForensics (referenced by Rinnan above) that focused on real-time event management as opposed to providing after-the-fact forensic analysis (Ex. 1006, p. 1). Chen also sought to improve on software like netForensics by providing log aggregation related to “security appliances, routers, and servers” as well as “performing data mining” on the aggregated data. (*Id.*).

67. Furthermore, a POSITA would have understood the value of storing such information in a common file or database, such as described here:

[4] defines normalization as the process of ***gathering individual security device data, and putting them into context that is easier to understand***, and claims that this enables us to map different messages about the same security event to a common alarm ID. [4] claims that normalization alone is a tremendous asset to security teams. Some commercial products use XML to structure the entries and a paper from GuardedNet [5] claims that ***normalization is a process that breaks an event into its component parts and places them into individual database fields***. The use of a database is also supported by Johan Beckers and Jean Paul Ballerini [11].
(Ex. 1014 p. 16, emphasis added).

B. System calls, hooking, and the Windows API

68. System calls are a mechanism by which user-mode processes can access system resources (e.g., the file system) to which the operating system wishes to control access. In

Windows, such kernel-level functions are not exposed directly to user-mode programs but are instead wrapped by the Windows API:

If you have ever examined ntdll.dll with QuickView, you might have noticed that it exports a set of functions that begin with the Nt prefix. These functions are actually small stubs of code that pass control to the Windows NT kernel (NTOSKRNL) using interrupt 2E. Many of the functions exported from kernel32.dll are nothing more than control transfer routines to the stubs located in ntdll. For example, when a Windows application issues a call to CreateFile located in kernel32.dll, the call is redirected to NtCreateFile, which passes it on to NT's kernel for further processing. The special design of this mechanism allows a device driver to hook these interfaces, thus providing a way for monitoring activities performed by Windows NT/2000 applications.

(Ex. 1018, p. 6).

69. A POSITA would have understood that in Windows, invoking system calls is done via API calls. Because of this structure, it is possible to intercept or monitor system calls (e.g., with a technique sometimes called “hooking”). This can be employed for purposes ranging from malicious (e.g., viruses, malware) to benign (e.g., performance monitoring, anti-malware).

API spying utilities are among the most powerful tools for exploring the inner structure of applications and operating systems. Unfortunately, neither the SDK nor the DDK provide any documentation or examples demonstrating a way for implementing such a utility. ***This article will attempt to shed some light on this subject by presenting several techniques for hooking API calls issued by Windows applications.***

There are several things you'll need to consider before sitting down to write your own API spying utility. First, you'll need to decide whether you want to spy upon a single application or set up a system-wide API interceptor. Each approach can be useful in different situations. For example, assume that you need to write an application that blocks the execution of specific processes according to rules set by an administrator. Obviously, you will need a way to monitor the execution of new processes and terminate the ones, which are marked as restricted. ***One way to accomplish that would be to establish a system-wide API interceptor that monitors calls made to the CreateProcess function (actually, to both the Ansi and Unicode versions of this function). Whenever an application issues a call to one of these functions in order to create a new process, your interceptor will gain control and perform whatever processing is necessary.***

(Id., p. 1, emphasis added).

70. This can be employed for purposes ranging from malicious (e.g., viruses, malware) to benign (e.g., performance profiling, policy enforcement, activity monitoring).

There are several things you'll need to consider before sitting down to write your own API spying utility. First, you'll need to decide whether you want to spy upon a single application or set up a system-wide API interceptor. Each approach can be useful in different situations. ***For example, assume that you need to write an application that blocks the execution of specific processes according to rules set by an administrator.*** Obviously, you will need a way to monitor the execution of new processes and terminate the ones, which are marked as restricted. One way to accomplish that would be to establish a system-wide API interceptor that monitors calls made to the CreateProcess function (actually, to both the Ansi and Unicode versions of this function). Whenever an application issues a call to one of these functions in order to create a new process, your interceptor will gain control and perform whatever processing is necessary.

Other types of applications might require a simpler API interceptor capable of monitoring just one application at a time. A good example is BoundsChecker from NuMega - a tool capable ***of analyzing API calls in order to detect memory leaks and other bugs lurking inside Windows applications.*** (*Id.*, p. 1, emphasis added).

71. One particular technique, “interrupt hooking,” allows drivers (e.g., device drivers) to be notified when an interrupt occurs that represents a user-mode process making calls to kernel-mode operations:

In the old days of DOS, interrupt hooking was widely used by TSR (Terminate and Stay Resident) utilities and other applications to extend the operating system functionality and monitor its behavior. Under Windows, interrupts still play a major role, and are mainly used as a portal that connects user-mode (a.k.a. ring 3) code to the operating system's kernel.

...

If you have ever examined ntdll.dll with QuickView, you might have noticed that it exports a set of functions that begin with the Nt prefix. These functions are actually small stubs of code that pass control to the Windows NT kernel (NTOSKRNL) using interrupt 2E. Many of the functions exported from kernel32.dll are nothing more than control transfer routines to the stubs located in ntdll. For example, when a Windows application issues a call to CreateFile located in kernel32.dll, the call is redirected to NtCreateFile, which passes it on to NT's kernel for further processing. The

special design of this mechanism allows a device driver to hook these interfaces, thus providing a way for ***monitoring activities performed by Windows NT/2000 applications.*** (*Id.*, p. 6, emphasis added).

C. Windows monitoring, kernel drivers, and APIs

72. There are various methods and technologies for monitoring events that occur on a Windows system.

73. The Windows software development kit includes a data structure called Win32_Process, which is part of the Windows Management Instrumentation:

The Win32_Process WMI class represents a sequence of events on a Windows® operating system. A descendent or member of this class is a sequence that consists of an interaction of one or more processors or interpreters, some executable code, and a set of inputs, for example, a client application running on a Windows system.
(Ex. 1016, p. 1).

74. Some data (called “properties”) available in this structure include “Name”, “ProcessId”, and “CommandLine”. (*Id.*, p. 1). In the Microsoft documentation, “Name” is described simply as “Label for an object.” (*Id.*, p. 4). “ProcessId” is described thus: “Global process identifier that you can use to identify a process. The value is valid from the time a process is created until it is terminated.” (*Id.*, p. 6). “CommandLine” is described as “Command line used to start a specific process, if applicable.” (*Id.*, p. 2).

75. A POSITA would have understood a “command line” in this context to mean the file name of the executable program and any arguments used when invoking the program. For example, the command line “notepad.exe my_document.txt” includes the file name of the executable (“notepad.exe”) and one argument (“my_document.txt”). This command line would start the Notepad program and cause it to open the file “my_document.txt” for editing.

76. Given a process identifier, it is possible to retrieve the command line for the process and thus learn the name of the executable file that is running in that process.

Processes are represented in WMI by instances of the Win32_Process class. This is derived from CIM_Process, which in turn is derived from CIM_LogicalElement, which we have met several times already. The primary key of a Win32_Process object is its Handle property; this value represents a numeric identifier that the operating system assigns to a process when it is created. Every process running on a computer at any given time has a different handle; the operating system uses this number to uniquely identify each process. Hence, you can easily retrieve a WMI object representing a process whose ID you know using code such as this:

```
Set refExistingProcess =  
GetObject("winmgmts:Win32_Process.Handle='1000'")  
(Ex. 1017, p. 3).
```

77. The data returned by this code will include the properties listed above (e.g., “Name” and “CommandLine”), from which can be learned the file name of the executable running in the process.

78. Some types of monitoring require “kernel-mode” drivers. A “kernel-mode” driver is able to run at the kernel level rather than at the user level.

As distinct from User mode drivers, Kernel mode drivers form part of the Windows NT Executive layer and run in Kernel mode, as their name implies. Kernel mode drivers are accessed and supported by the I/O Manager. Requests are sent to the I/O Manager as a result of executing an I/O system service, such as a Win32 CreateFile () or ReadFile () function, or a Windows NT native function such as NtCreateFile () or NtReadFile ().
(Ex 1010, p. 226).

79. Kernel-mode drivers come in various forms including “file system drivers,” “intermediate drivers,” “device drivers,” and “mini drivers”. (Ex 1010, p. 226).

80. Monitoring network traffic, for example, typically requires some kind of device driver. Windows offers interfaces that enable kernel drivers to interface with the system and obtain the data. One such interface is called the “Transport Driver Interface,” or TDI. TDI drivers are intermediate drivers (which are kernel drivers as explained above).

Networking protocols, or "TDI drivers" as they are called, are in fact relatively standard Windows NT intermediate layer drivers. They implement a standard upper edge, as defined by the TDI specification. (Ex 1010, p. 585).

81. TDI drivers can be used to monitor networking. A well-known example is the program "TDImon" (short for TDI monitor).

The Network Monitor is implemented based on TDIMon [19] and WinDump [34] which requires WinPcap [35]. TDIMon monitors activity at the Transport Driver Interface (TDI) level of networking operations in the operating system kernel. It can capture all network events associated with process information (Ex. 1012, p. 5).

82. Another example of a kernel-level monitoring module is APISpy32. This tool monitors at the kernel level for system calls:

For example, APISpy32 [10] monitors the system from the kernel level by recording the system calls executed by the system. WinDump monitors the system from the network level. The rest of our architecture consists of several Perl scripts used in the correlation process. (Ex. 1013, p. 6).

83. A POSITA would have understood that tools that monitor the kernel for system calls, like APISpy32, would require kernel-level access.

D. Behavior monitoring techniques for anti-malware

84. Many techniques for detecting and responding to malware existed at the time of the alleged invention. These included signature scanning (looking for particular sequences in the content of a suspicious file) and integrity checking (using cryptography to verify that a file is intact and correct).

85. Signature-based virus scanning, for example, is described as follows in the 2003 textbook, "Malware: Fighting Malicious Code":

One of the simplest and most popular ways in which antivirus software detects malicious code is through the use of virus signatures. The antivirus

vendors collect malware specimens and “fingerprint” them. Thousands of signatures are gathered together in a database for use in an antivirus scanner. The database of such signatures is distributed to systems that require protection. When scanning files for malicious code, antivirus software compares the current file to its signature set and determines whether the file matches a signature of a known malware specimen. (Ex. 1015, pp. 8-9).

86. However, this technique has limitations, namely around the timely creation and propagation of signature databases:

One of the biggest challenges to this signature-based method of malware detection is that antivirus software needs to include a signature for the virus to discover it on the victim's system. This means that antivirus vendors strive to collect new virus samples, develop patterns that fingerprint them, and distribute signature updates to the customers as quickly as possible. (Ex. 1015, p. 9).

Still, even with rapid and frequent updates, one of the weaknesses of the signature-matching approach is that it is always playing catch-up with malware authors as they release brand new, or even slightly modified, specimens into the wild. Additionally, a bad guy might create a custom virus, keeping it close to the vest before releasing it against a particular target. Without a widespread release, antivirus software developers cannot create a signature until the virus has been deployed against its target, which might be too late to stop major damage. Another significant disadvantage of this signature-based detection technique is that, in its pure form, it cannot identify malicious code designed to automatically change itself as it propagates, thereby modifying itself so that it doesn't match any signatures. (Ex. 1015, pp. 10-11).

87. Integrity scanning works in the following manner:

When defending against viruses, we are dealing with creatures that modify their host programs as they spread. Therefore, one way to detect the presence of a virus is to discover files that have been unexpectedly modified. The integrity verification process aims to achieve this goal by following these steps:

1. While the machine is in a pristine state, compute fingerprints (in the form of checksums or cryptographic hashes) of files that need to be monitored, and record them in a baseline database.
2. When scanning the file system for suspicious modifications, compute fingerprints of monitored files and compare the values to those in the baseline.

3. If unexplained differences between the current state and the baseline are detected, issue an alert.
(Ex. 1015, p. 13).

88. Another technique is behavior monitoring. Recognizing the limitations of signatures, as described above, this technique aims instead to use heuristics or patterns of behavior to detect malware. Such malware, even if it has never been seen before and does not have a known signature, may nevertheless exhibit certain behaviors characteristic of malware, and it can be detected on the basis of matching those patterns of behavior with a “heuristics-based detection engine.” For example, Skoudis describes behavior monitoring using emulation:

Realizing the limitations of signature-based detection methods, antivirus vendors have devised similar ways in which *they can detect previously unseen viruses that exhibit certain behavioral and structural characteristics*. Symantec, for instance, calls this feature of its Norton AntiVirus product Bloodhound. A heuristics-based detection engine scans the file for features frequently seen in viruses, such as these:

- Attempts to access the boot sector.
- Attempts to locate all documents in a current directory.
- Attempts to write to an EXE file.
- Attempts to delete hard drive contents.

As the heuristics scanner examines the file, it usually assigns a weight to each virus-like feature it encounters. If the file's total weight exceeds a certain threshold, then the scanner considers it malicious code. If the scanner's developer sets the threshold too low, then the user could be overwhelmed with false alarms. On the other hand, if the threshold is set too high, or if virus-like features are not properly identified, then the detector will miss too many viruses. Either way, the user's protection is limited unless the sensitivity is set just right.

This technique would not be very helpful if antivirus software was able to detect malware only after the virus exhibited malicious behavior such as infecting programs or deleting files. If that were the case, you might get a warning from the antivirus software that says, “Your system has just been completely undermined by a virus! Have a nice day.” Although this is certainly interesting information, you need to get the warning before the malware has its way with your machine. The trick is to parse the suspicious file in a way that allows antivirus software to estimate what actions would be performed if the virus actually has a chance to execute. This analysis

must occur before the code runs. Antivirus software accomplishes this goal by attempting to emulate the processor that would have executed the potentially malicious program. In the case of executables compiled for Intel x86 machines, this approach calls for emulating key features of the x86 processor. In the case of VBScript macros embedded into Microsoft Office documents, this approach requires emulating basic functionality of the VBScript processing engine.
(Ex. 1015, pp. 11-12, emphasis added).

89. As illustrated above, a POSITA would have understood that a “heuristic” is a pattern, rule, or other guideline that is used to make a good estimate about classification. So, for example, a virus signature, is usually **not** a heuristic. It is meant to definitively classify a test file as either a specific virus or not that specific virus. On the other hand, rules like those described above that relate to “virus-like features” attempt to estimate whether or not a test file is a virus or not a virus. A POSITA would have known that a heuristic is not tied specifically to virus detection but to any kind of classification system.

90. Also described is a technique called “behavior blocking”, which allows suspicious code to execute until it attempts to perform actions that identify it as malicious:

Instead of relying on signatures to detect known malware specimens, ***behavior blockers keep an eye out for suspicious actions to recognize and block behavior typically associated with malicious code.*** For example, attempts to perform the following actions could suggest that the program is malicious:

- Writing to sensitive portions of the registry
 - Creating a file on a system directory
 - Overwriting browser settings
 - Adding a browser plug-in
- (Ex. 1015, p. 24, emphasis added).

91. Indeed, tools to do such behavior blocking were commercially available at the time:

One tool that uses a behavior-based technique to protect workstations against malicious mobile code is Finjan SurfinGuard, which you can purchase from www.finjan.com. SurfinGuard runs in the background on your machine, creating a tight sandbox around programs and browser scripts that would otherwise have unrestrained access to the system's

resources. For example, Figure 4.18 shows a warning that SurfinGuard popped up when a malicious site managed to push a malicious ActiveX control to a system in my lab. The control slipped through Internet Explorer's security settings because I purposefully misconfigured them—this resembles an environment where a careless user agreed to install an untrusted ActiveX control, or where the browser contained an unpatched vulnerability.

My browser downloaded the ActiveX control and attempted to execute a program named Loader.exe. Fortunately, SurfinGuard alerted me about this activity, giving me an option to block the action or to continue monitoring the suspicious executable. Without this layer of protection, my slightly misconfigured workstation would fall victim to the malicious ActiveX control. Similar behavior-blocking functionality exists as part of the Tiny Personal Firewall (TPF) product sold by Tiny Software at www.tinyspftware.com. TPF allows the user to categorize programs into groups based on their trustworthiness, with each group subject to different restrictions. When protecting the misconfigured machine that I described in the previous example, TPF offered to execute the suspicious program within a sandbox appropriate for the Restricted Applications group, as you can see in Figure 4.19.

Behavior-monitoring tools such as SurfinGuard and TPF help protect workstations against malicious mobile code, because they can often recognize a new malware specimen without relying on signatures that might not even exist for many mobile specimens. In addition to supplementing traditional antivirus software, behavior monitors play well with tools that specifically target spyware programs.
(Ex. 1015, pp. 24-26).

92. Screenshots of the cited figures 4.18 and 4.19 are reproduced hereinbelow:



(Ex. 1015, p. 25, Figure 4.18).



(Ex. 1015, p. 26, Figure 4.19).

93. The work of researchers such as Forrest, Hofmeyr, and Somayaji in the 1990's proposed immunological approaches to malware detection and prevention (i.e., designing systems that function similarly to the human immune system, in which native entities and behaviors are tolerated while foreign pathogens are not), in which normal and abnormal behavior models (which they call a "behavioral signature") can be developed over time and allow for detecting "foreign" (i.e., emergent) threats.

Do the normal databases allow the system to discriminate between normal and abnormal behavior? To date, we have constructed databases of normal behavior for three Unix programs: sendmail, wu.ftpd (a Linux version of ftpd), and lpr. When comparing the normal database for one program (e.g., sendmail) with traces of normal behavior of a different program (e.g., ls), we observed 40%–80% mismatches over the length of the foreign (e.g., ls) trace. We also observed clear detection of several common intrusions for the three programs (mismatch rates generally ranged from 1%–20% for the length of the trace). ***These results suggest that short sequences of system calls do provide a compact signature for normal behavior and that the signature has a high probability of being perturbed during intrusions.***

Although this method does not provide a cryptographically strong or completely reliable discriminator between normal and abnormal behavior, it is much simpler than other proposed methods and could potentially provide a lightweight, real-time tool for continuously checking executing code.

(Ex. 1019, p. 6, emphasis added).

94. Thus, it was well understood in the art that signature scanning and integrity checking, although useful tools, were not the only anti-malware techniques and that they had limitations (e.g., inability to detect novel malware for which no signature is available). It was also well understood in the art that certain behaviors tend to indicate the presence of malware, and that such behaviors can be monitored or even blocked in order to detect and/or prevent malware.

E. Malware and Privilege Escalation

95. A POSITA would have known that malware, such as worms and viruses, often use "privilege escalation," wherein the program starts out running with lower privileges (e.g., "user"

privileges) and through unauthorized means obtains higher privileges (e.g., “super user”, “root”, or “administrator” privileges).

It is already common for viruses to disable anti-virus software, while the egress protection of personal firewalls can be corrupted by Trojans overwriting the memory space of authorized programs. Escalating from limited access to super user access is often quite easy, as the vulnerabilities leveraged are usually not characterized as “critical” and often go unpatched. In other cases, including the Code Red II [3] and Blaster [16] worms, the exploit gains full system-level access directly, as the targeted service is already running with full system privileges. (Ex. 1011, p. 4).

96. A POSITA would have also known that malware typically wants “elevated” privileges in order to get past host system defenses, bypass anti-virus, or otherwise pierce defenses and/or remain undetected.

VI. OVERVIEW OF THE CHALLENGED PATENT

A. THE ‘244 PATENT SPECIFICATION AND CLAIMS

97. U.S. Patent No. 8,181,244 to Matthew L. Boney (“the ‘244 patent”) is titled “Backward Researching Time Stamped Events to Find an Origin of Pestware” and is directed to identifying the source of pestware events on a computer occurring near a particular time:

The present invention can provide a system and method for identifying an origin of suspected pestware activity on a computer. One embodiment includes establishing a time of interest relating to a suspicion of pestware on the computer, identifying, based upon the time of interest, indicia of pestware and accessing at least a portion of a recorded history of sources that the computer received files from so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of a source that is suspected of originating pestware. (Ex. 1001, 2:17-26).

98. In the alleged invention, an activity monitor logs activity occurring on the computer, for example by monitoring API calls made by processes.

The activity monitor 116 in several embodiments monitors activities on the computer and stores information about the activities in the activity log 121

so as to assist the research module 108 in identifying activities associated with pestware and the source of the pestware.
(Ex. 1001, 5:21-25).

In other embodiments, the activity monitor 116 is realized by a kernel-mode driver that may be loaded during a boot sequence for the protected computer or anytime later. In these embodiments, the activity monitor 116 is configured to log API calls in the activity log 121. In many variations for example, when a process (e.g., the pestware process) attempts to spawn another pestware process or alter a registry entry, the API call utilized by the process is intercepted before it is carried out by an operating system of the protected computer. In this way, the attempt may be logged in the activity log 121 and the process may be analyzed to determine whether it is known to be a pestware-related process.
(Ex. 1001, 5:41-52).

99. Thereafter, a user can investigate suspected malicious activity by specifying a time of interest and causing the research module to access logged information to look for indications of pestware around that timestamp:

As shown, in this embodiment a time of interest is initially established based upon a suspicion that pestware has infected a computer (Block 502). In some embodiments, for example, a user may establish the time of interest based upon events the user observed (e.g., pop-ups or a system crash).
(Ex. 1001, 8:59-63).

As shown in FIG. 5, once a time of interest is established, suspicious activity is identified on the computer based upon the time of interest (Block 504). The time of interest may be established, for example, to include a time period before the time stamp is issued so that a search for suspicious activity is limited to a particular time period. In some embodiments, the activity log 120 and/or one or more of the N files 130 are accessed and analyzed to determine whether any activity during the time of interest is suspicious (e.g., the activity indicates in some way that it may be associated with pest ware).
(Ex. 1001, 9:9-19).

100. Once suspicious activities are found, processes or files can be identified that are associated with that activity:

As depicted in FIG. 5, once one or more activities are identified as being suspicious (Block 504), one or more objects (e.g., processes or files) on the computer are associated with the suspicious activity (Block 506). For example, the research module 108 may search the activity log 121 and/or

one or more of the N files for information that associates the suspicious activity to one or more processes and the processes may be related to one or more files.

(Ex. 1001, 9:37-44).

101. Thereafter, the source of the objects may be determined and reported to a research entity:

As shown in FIG. 5, recorded information on the computer is then traversed in order to trace to an origin of one or more of the objects (Block 508). For example, a search of one or more of the N files 130 (e.g., the cache index 243) may be carried out to identify the source of a suspicious process and/or files. Once the source of the suspicious object(s) is identified (Block 510), the source is then reported (e.g., to a remote research entity)(Block 512). (Ex. 1001, 9:45-52).

102. Sources of potential pestware thus reported may be identified by, e.g., their I.P. address or URL:

For example, the identities (e.g., I.P. address, URL, email client or program name) of sources (e.g., web sites, email or program) of the pestware may be collected in a centralized data store (not shown) and then subsequently reported to other users. (Ex. 1001, 3:41-45).

B. PROSECUTION HISTORY

103. I understand that the application that issued as the '244 patent, U.S. Patent Application No. 11/408,145, was filed on 4/20/06. (Ex. 1001, Cover Page).

104. Prosecution began with a non-final Office Action dated 6/22/09. The final round of rejections and amendments is illustrative of the Examiner's findings during prosecution.

105. I further understand that in the non-final Office Action that issued on 6/22/09, claims 1-23 were rejected. Claims 1-23 were rejected under 35 U.S.C. § 103(a) as unpatentable over U.S. Patent Application No. 2004/0024864 to Porras et al. ("Porras") in view of U.S. Patent Application No. 2005/0268112 to Wang et al. ("Wang").

106. It is my understanding that Applicants responded to the non-final Office Action by amending the independent claims. Specifically, the independent claim 1 was amended to recite, in part “computer, the method comprising: receiving, from a user, establishing”, and independent claim 12 was amended to recite, in part “a computer, including the system comprising:...to detected activity activity on the computer that is indicative of pestwarea suspicion of pestware on the compuer...at least one log on a file storage device of the computer”. (Ex. 1002, pp. 193-95, underlines showing the amended language). In associated remarks, Applicants argued that claim 1 uses a user-supplied “time of interest” to guide an automated (computer-implemented) process for researching and identifying the source, and the cited references do not disclose that claim 1 feature. (Ex. 1002, pp. 198-199). Applicants further that the references do not recite the claim 12 feature of “a timestamp module configured to generate a time of interest in response to detected activity on the activity on the computer that is indicative of pestware.” (Ex. 1002, p. 199).

107. A final Office Action was submitted, issued 2/22/10, that rejected claims 1-6, 8, 10-16, and 19-23. Ex. 1002, p. 162. The final Office Action rejected claims 1-6, 8, 10-16, and 19-23 under 35 U.S.C. § 103(a) as unpatentable over Porras, Wang, and further in view of U.S. Patent Application No. 2004/0225877 to Huang.

108. Applicants responded to the final Office Action by submitting a Request for Continued Examination and amending independent claim 1 to recite, in part “externally networked sources that the computer received files...an externally networked source that is suspected of originating pestware”, and independent claim 12 to recite, in part “externally networked source of potential pestware activity on the computer...externally networked source of potential pestware activity”. (Ex. 1002, pp.147-49). Applicants contended that the combination of references do not teach the above added limitations to claims 1 and 12. (Ex. 1002, pp. 153-54).

109. In the non-final Office Action, issued 9/10/10, claims 1-6, 8, 10-16, and 19-23 were rejected. (Ex. 1002, p. 123). The non-final Office Action rejected claims 1-6, 8, 10-16, and 19-23 under 35 U.S.C. § 103(a) as unpatentable over Porras and U.S. Patent Application No. 2007/0168694 to Maddaloni et al. (“Maddaloni”).

110. Applicants responded to the non-final Office Action by not amending any of the claims and arguing that the Maddaloni reference is disqualified as prior art under 35 U.S.C. § 103(c)(1). (Ex. 1002, p. 115).

111. In the non-final Office Action, issued 3/28/11, claims 1-6, 8, 10-16, and 19-23 were rejected. Ex. 1002, p. 87. The non-final Office Action rejected claims 1-6, 8, 10-16, and 19-23 under 35 U.S.C. § 103(a) as unpatentable over Porras and U.S. Patent Application No. 2007/0094725 to Borders. (Ex. 1002, p. 89).

112. Applicants responded to the non-final Office Action by amending claim 1 to recite, in part “receiving, from a user via an interface of the computer...accessing, using a hardware processor of the computer,...wherein the recorded history of externally networked sources is stored on a file storage device”, and amending claim 12 to recite, in part “ a hardware processor; a file storage device coupled to the hardware processor;” a timestamp, research portion, and reporting portion “to be executed by the hardware processor”. (Ex. 1002, 65-67).

113. Allowance of the '244 patent followed. The Notice of Allowability, issued 1/23/12, indicated that an Examiner Interview occurred on 12/20/11 and Applicants agreed, via Examiner Amendment, to amend independent claims 1 and 12 to recite the limitations of claims 5, 19, 21 and add the limitations “issuing a timestamp after receiving the time of interest to initiate the pestware research”. (Ex. 1002, p. 36). Claims 1-4, 6, 8, 10-16, 20, 22, and 23 were deemed allowed.

VII. LEVEL OF SKILL IN THE ART

114. I understand that Petitioners submit that a person of ordinary skill in the art (“POSITA”) at the time of the earliest possible date for the ‘244 patent would have had a bachelor’s degree in computer science, or a related field, or an equivalent technical degree or equivalent work experience, and an additional one to two years of education or experience in the field of computer security topics such as malware/exploit detection. More education can supplement practical experience and vice versa.

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

VIII. OVERVIEW OF THE PRIOR ART

A. OVERVIEW OF CHEN

116. “ECF – Event Correlation for Forensics” is a 2003 research paper by Kevin Chen et al. Its focus is consolidating and correlating information from various log sources:

The focus of the research described in this paper is on the nature of the event information provided in commonly available computer and other logs and the extent to which it is possible to correlate such event information despite its heterogeneous nature and origins. The strategic purpose of the research has been to develop a means by which a consolidated repository of such information can be constituted and then queried in order to provide an investigator with post hoc event correlation for forensics purposes (ECF). The paper provides an account of the log processing techniques utilized, and the nature of the database and query engine that have been developed in our current prototype and two examples of scenarios investigated and identified by the current prototype.
(Ex. 1006, p. 1).

117. Chen discloses capability of ingesting logs in the several formats (“Apache Server Log”, “Windows 2000 Security Log”, “Door Log”, “Browser Cache Logs”, and “*NIX SYSLOG – POP3, SSH, sendmail, smap”). (Ex. 1006, pp. 4-5). Chen also anticipates additional types of logs being used in its system: “It is expected that future parsers will likewise be regular expression

based and we are reworking some of the other parsers to use the same approach. This will facilitate extensibility with regard to accommodating new logs or re-formatted versions of existing logs.” (Ex. 1006, p. 50).

118. Chen also indicates that additional logs may be helpful. For this reason, the system is meant to be extensible so that additional data can be incorporated easily.

An overriding objective in the research and development has been that of extensibility: it must be relatively straightforward to incorporate new types of event and transactional logs and to accommodate log changes and configurable log formats that may vary from system to system and organization to organization. The ECF system must provide an extensible framework for the analysis and correlation of heterogeneous event and transaction records.
(Ex. 1006, p. 2).

The results presented in Section 4 above indicate that cross-correlation of event information emanating from heterogeneous logs potentially provides a powerful basis for the forensic identification of scenarios of interest even though only a very small base of common denominator information is being taken into account. Clearly, the inclusion of additional information from various log sources can only expand the potential usefulness of the technique.
(Ex. 1006, p. 9).

119. Of particular interest in Chen’s disclosures is having a “canonical representation” of events in its database that includes time information and source information:

The decision was made to opt for the former viz., for a canonical representation of what is essentially the minimal or base set of fields which appear in all event records viz., Time, Subject, Action, and Object. These fields map intuitively to almost all event records no matter how varied the events or systems on which they occur, and so a single database table (the Canonical table) which represents that information is consequently the central information store used for event correlation in ECF.
(Ex. 1006, p. 2).

120. Chen discloses a user interface for querying the stored activity information. It allows the user to specify timestamps and observe correlations among the data:

The investigator would presumably be interested in knowing times when the administrator account on the workstation was logged in to. ***The ECF tool Dynamic Query interface was used to generate the following query*** which will return a list of records relating to logins to the ‘administrator’ account on the workstation in question (‘dsto’)...

This query returned a single record from our test logs which we do not show here due to space limitations. ***However, the record returned occurred at the time ‘2003-08-08 04:56:5’ (UCT).*** The investigator would also be interested in other accounts that had been used to login to the workstation at around the same time. Therefore, in a similar fashion to the previous scenario ***we perform a query to determine all the users who logged onto the workstation ‘dsto’ in the time window commencing ten minutes prior to the identified records,*** and finishing ten minutes after it. The following SQL query, which was generated using the Dynamic Query interface, extracts the desired records:

```
SELECT      *      FROM      CANONICAL,      LOGRECORD      WHERE
CANONICAL.LOGID      =      LOGRECORD.LOGID      AND
CANONICAL.TIME      >=      '2003-08-08      04:46:51'      AND
CANONICAL.TIME      <=      '2003-08-08      05:06:51'      AND
CANONICAL.ACTION = 'Logon' AND LOGRECORD.LOGNODE =
'dsto';
```

(Ex. 1006, p. 8, emphasis added).

121. Chen also anticipates the ability to automate additional parts of its process:

Note that it is our intention that the ECF tool will eventually be able to make many of the inferences that we currently perform manually, by itself. It is envisaged that data mining approaches, combined with the ability to relate the additional log-specific information between the different log types, will allow the tool to extract only the most relevant logs, leaving extraneous ones out. This problem is analogous to the current problem with intrusion detection systems which can generate very large amounts of alerts, many of which are not relevant.

(Ex. 1006, p. 9).

B. OVERVIEW OF PICCARD

122. U.S. Patent Publication No. 2007/0006310 to Paul L. Piccard, “Systems and Methods for Identifying Malware Distribution Sites” (“Piccard”), is directed toward identifying the source of malware that has been detected. It does so through the use of several modules. The

“malware detection module” is responsible for identifying a suspicious file. Piccard describes several exemplary ways this can be done, including by monitoring for suspicious activity patterns:

The first operation illustrated in FIG. 2 is to analyze a file to determine that the file includes potential malware (block 200). In the illustrated embodiment, a malware detection module (e.g., the malware detection module 118) scans files of a protected computer (e.g., the protected computer 102) to locate the file. The malware detection module can scan files of the protected computer on a periodic or some other basis. Alternatively, or in conjunction, operation of the malware detection module can be triggered based on determining that the file is being downloaded or has been downloaded using a Web browser (e.g., the Web browser 114).

In the illustrated embodiment, the malware detection module compares the file with a set of malware definitions to determine if the file includes potential malware. The set of malware definitions can include representations of malware, suspicious activities that are indicative of or that are common to malware, or both. For example, the set of malware definitions can include a hash value or a digital signature of malware. Such as one that is generated using Message Digest 5 (“MD5”). In this example, the malware detection module generates a hash value for the file and compares the hash value of the file with a set of hash values of malware to determine whether there is a sufficient match. As another example, the set of malware definitions can include a Cyclical Redundancy Code (“CRC”) of a portion of malware. In this example, the malware detection module generates a CRC for the file and compares the CRC of the file with a set of CRCs of malware to determine whether there is a sufficient match. ***As a further example, the set of malware definitions can include suspicious activities related to third-party cookies or related to entries or modifications of registry files of an operating system.*** (Ex. 1007, [0019]-[0020], emphasis added).

123. Once a suspicious file has been identified, the “Web site identification module” is responsible for determining the source of the file. The browser history log contains information about the source of downloaded files:

As illustrated in FIG. 1, the memory 112 also stores a history log 116, which is maintained by the Web browser 114 to provide a record of browsing events. In particular, when a file is accessed and downloaded, the Web browser 114 records a Web address of the file in the history log 116. A Web address typically specifies a location of a file within a Web site. For example, a Web address can be a Uniform Resource Identifier (“URI”) of a file, such as a Uniform Resource Locator (“URL”) of the file. It is also

contemplated that a Web address can be defined in various other ways, such as using an Internet Protocol (“IP”) address or any other identifier of a source of a file.
(Ex. 1007, [0013]).

124. Using this information, the “Web site identification module” can identify the source of the suspicious file:

The second operation illustrated in FIG. 2 is to search a download history log to identify a Web site from which the file was downloaded (block 202). In the illustrated embodiment, once the malware detection module determines that the file includes potential malware, a Web site identification module (e.g., the Web site identification module 120) accesses the download history log to identify the Web site from which the file was downloaded. The download history log serves to provide a record of downloading events. For example, the download history log can be a Web browser's history log (e.g., the history log 116). In this example, the Web site identification module can access the Web browser's history log to identify a Web address of the file. As described previously, the Web address of the file can be a URL of the file, which can have the following format: `http://www.DomainName.com/Subdirectory/FileName.html`, where “`http://`” specifies a communication protocol used to download the file, “`www.DomainName`” specifies a domain name of the Web site from which the file was downloaded, “`/Subdirectory/`” specifies a subdirectory within the Web site from which the file was downloaded, and “`FileName.html`” specifies a name of the file. ***Thus, by searching the Web browser's history log using the name of the file, the Web site identification module can identify the Web site from which the file was downloaded, such as in terms of the domain name of the Web site.***
(Ex. 1007, [0021], emphasis added).

125. Thereafter, Piccard teaches a “reporting module” responsible for notifying a “remotely-located computer” of the website, which may be a malware distribution site, in order to prompt further research about the website:

The third operation illustrated in FIG. 2 is to report that the Web site is a potential malware distribution site (block 204). In the illustrated embodiment, once the Web site identification module identifies the Web site, ***a reporting module (e.g., the reporting module 122) reports information regarding the Web site to a remotely-located computer that is connected to the protected computer.*** This information can identify the Web site as a potential malware distribution site, such as in terms of the domain name of the Web site. Alternatively, or in conjunction, this

information can identify the file as including potential malware, such as in terms of the name of the file or the URL of the file. It is also contemplated that this information can include a representation of the file or can identify suspicious activities related to the file. ***This information as well as any additional relevant information can be analyzed at the remotely-located computer to determine whether the file does, in fact, include malware and whether the Web site is, in fact, a malware distribution site.*** If the Web site is determined to be a malware distribution site, a new or updated set of malware definitions can be generated based on content within the Web site, and the new or updated set of malware definitions can be provided to the protected computer. In addition, content within the Web site can be monitored on a periodic or other basis for new or updated malware. Also, a new or updated list of malware distribution sites can be generated so as to identify the Web site, and the new or updated list of malware distribution sites can be provided to the protected computer.
(Ex. 1007, [0023], emphasis added).

C. OVERVIEW OF LI

126. “UCLog: A Unified, Correlated Logging Architecture for Intrusion Detection” is a 2004 research paper by Zhenmin Li et al. It is directed to demonstrating methods for correlating different types of logs in order to improve intrusion detection:

Activity logs can be used for intrusion detection; however, most previous work on intrusion detection examines only activity logs from a single component. Doing so fails to take advantage of the naturally existing correlations among activities in different types of logs, such as network logs and system call logs.

This paper explores correlation for intrusion detection. Specifically, we propose UCLog, a unified logging architecture that can effectively capture correlations among entries in different types of logs. UCLog enables the intrusion detection system to make some sense of the myriad of different available logs and correlate the information the logs present to enhance the intrusion detection process.
(Ex. 1008, p. 1).

127. The architecture of UCLog involves several components. Of particular importance are the loggers, monitors, activity database, and rule generator. Figure 2 illustrates the interaction of these components:

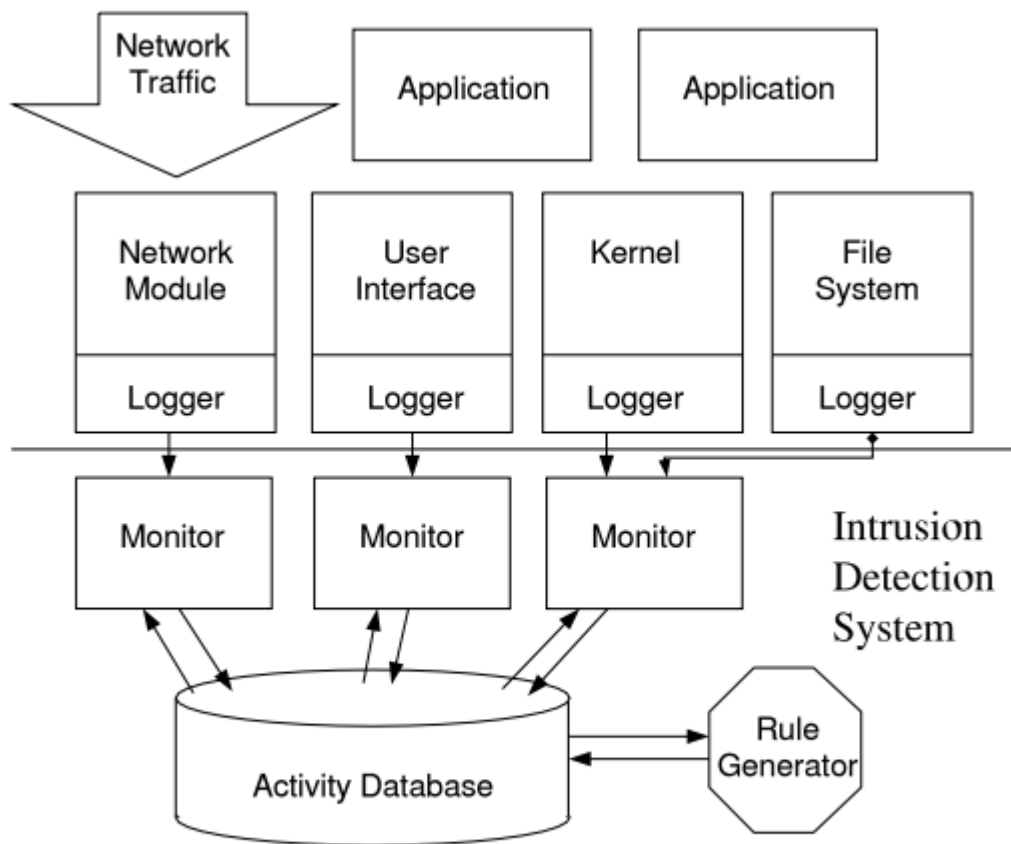


Figure 2. UCLog architecture

(Ex. 1008, p. 4, Figure 2).

128. The various components are described as follows:

Loggers collect information about various aspects of system behavior and send this data to *monitors*. *Monitors* watch [sic] system behavior and collaborate with each other through the centralized *activity database*. *Monitors* filter and compress audit data collected by *loggers*, and compare this data with existing models or signatures. The centralized activity database stores and manages audit data in an organized way. It also supports queries from a monitor to find out correlated activities watched by other monitors, or to trace to the specific processes or users that directly or indirectly cause the occurring intrusion. The *rule generator* produces the models for normal or abnormal behavior and signatures for previous intrusions. The rule generator can run either in the background or off-line, and either on the same or possibly several other machines. (Ex. 1008, p. 4, emphasis in original).

129. Li discloses various loggers that record many types of activity, including system calls and network traffic:

Kernel API Logger: A kernel API logger should record system calls, the parameters of those calls, and the time at which they are made.

Network Logger: A network logger could record all packets passing through the system, all packet headers received by the system, or even record higher, summary-level information (e.g. Cisco NetFlows [10]). The actual implementation would depend on the desired circumstances.

File-System Logger: A file-system logger might detect changes to significant blocks, such as the password file, or detect unusual access patterns. For example, a process which attempts to touch every block once could be harmless, such as a disk defragmentation process, or it could be some user attempting to illegally duplicate the disk.

User Behavior Logger: It would be useful to have some module that records a user's interactions with the system, such as when each user logs in and out.

System Events Logger: A facility that monitors the children, files, and directories created by some process would be useful in many situations.

Application-Specific Logger: Additional application-specific loggers can be added as needed.
(Ex. 1008, p. 5).

130. Information from these loggers is assembled in the activity database, in a standardized format that includes process identification and timestamps for efficient querying and analysis:

The activity database stores and manages audit data in a very organized way. It is implemented as an ***efficient relational database management system in order to present monitors with an interface to query the database to find correlated activities or to trace to the root and the path of an intrusion.*** To prevent intrusions from breaking into the activity database, the activity database should run in a protected environment and should support only a secured query interface to monitors and the rule generator.

The activity database consists of a set of tables, each corresponding to the filtered log entries from a monitor. Each table describes some measurable and distinct aspect of system behavior. For example, in our prototype

implementation, there is a table of system calls which records <**processID**, processName, systemCall, date, **timeStamp**, etc>, a table for network connections recording <**startTime**, duration, sourceHost, destHost, bytesFromSource, bytesFromDest, **processId**, etc>, and another for system behavior recording <**date**, **time**, eventCode, **pid**, user, **parentID**>. (Ex. 1008, p. 6, emphasis added).

131. The abilities to both identify the source of the activity and to operate queries based on timestamps are of particular importance in Li's disclosures:

Another key advantage of the UCLog architecture is the ability to trace back and identify the source and the execution path of the occurring intrusion. At an intrusion, a system administrator who is assigned to analyze the intrusion might ask questions like "What process was running when a large spike of network activity occurred yesterday at midnight?". In most current systems, the administrator needs to examine the logs manually in order to find out the answers to his/her questions. But in UCLog, the activity database supports various queries to provide answers to these questions. To further simplify the administrator's job, some simple tools can be built on top of the activity database with a user-friendly interface and relieve the administrator from writing SQL statements. (Ex. 1008, p. 7).

132. In addition to storing the collected activity data, Li's disclosures also include the creation and application of rules to categorize "appropriate or illegal system behavior":

In addition to the output of monitors, the activity database also stores "rules" (models and signatures) to describe appropriate or illegal system behavior. Since data can be retrieved in so many different ways from the activity database, it is possible to store several different granularities of a rule. In general, the most commonly used rule set would apply to any system-level behaviors; however, the activity database can also store rules describing how a particular instance (e.g., a user, an application or a file) should behave. For example, the activity database can store rules describing the network traffic generated by FTP or the SMTP daemon. Although instance-specific rules would be too expensive to use when filtering all the data streams, they would be of invaluable use in identifying (or ruling out) potential causes of anomalies. (Ex. 1008, p. 7).

133. Li's experiments revealed that UCLog's application of correlations increased detection accuracy and decreased false positives:

Our results show that UCLog can significantly improve the accuracy of intrusion detection. In abnormal runs, UCLog can improve the confidence level of abnormal behavior. In normal runs, UCLog can decrease the number of false positives. Moreover, UCLog is relatively efficient and has reasonable space requirements.
(Ex. 1008, p. 10).

IX. GROUND 1: CLAIMS 1-16 ARE RENDERED OBVIOUS BY LI IN VIEW OF PICCARD

134. As I describe in greater detail hereinbelow, a POSITA would have found Claims 1-16 rendered obvious by Li in view of Piccard.

A. Motivation to Combine Li and Piccard

135. A POSITA would have been motivated to combine the teachings of Li and Piccard. As I discussed in Section VIII.C., Li is directed toward methods for correlating different types of logs in order to improve intrusion detection:

Activity logs can be used for intrusion detection; however, most previous work on intrusion de4tection examines only activity logs from a single component. Doing so fails to take advantage of the naturally existing correlations among activities in different types of logs, such as network logs and system call logs.

This paper explores correlation for intrusion detection. Specifically, we propose UCLog, a unified logging architecture that can effectively capture correlations among entries in different types of logs. UCLog enables the intrusion detection system to make some sense of the myriad of different available logs and correlate the information the logs present to enhance the intrusion detection process.
(Ex. 1008, p. 1).

136. Li's experiments revealed that UCLog's application of correlations increased detection accuracy and decreased false positives:

Our results show that UCLog can significantly improve the accuracy of intrusion detection. In abnormal runs, UCLog can improve the confidence level of abnormal behavior. In normal runs, UCLog can decrease the number of false positives. Moreover, UCLog is relatively efficient and has reasonable space requirements.
(Ex. 1008, p. 10).

137. Thus, Li teaches using activity information from multiple sources including system calls and network traffic:

Kernel API Logger: A kernel API logger should record system calls, the parameters of those calls, and the time at which they are made.

Network Logger: A network logger could record all packets passing through the system, all packet headers received by the system, or even record higher, summary-level information (e.g. Cisco NetFlows [10]). The actual implementation would depend on the desired circumstances.

File-System Logger: A file-system logger might detect changes to significant blocks, such as the password file, or detect unusual access patterns. For example, a process which attempts to touch every block once could be harmless, such as a disk defragmentation process, or it could be some user attempting to illegally duplicate the disk.

User Behavior Logger: It would be useful to have some module that records a user's interactions with the system, such as when each user logs in and out.

System Events Logger: A facility that monitors the children, files, and directories created by some process would be useful in many situations.

Application-Specific Logger: Additional application-specific loggers can be added as needed.
(Ex. 1008, p. 5).

138. Li also explains that these logging components can be distributed across a whole system rather than existing on one machine alone. Distributed analysis is preferred:

Each of the types of loggers listed can collect host-specific or systemwide information. The latter are distributed loggers. Distributed loggers are very useful and can be implemented to log system-wide information. For example, it is very common for organizations to have dedicated servers that collect and store logs obtained from different sources (e.g. syslogs obtained from each host in the organization). Obtaining a system-wide view is very helpful for intrusion detection, as related malicious activities may appear in several hosts. Correlating this information can help detect attacks that otherwise may not be obvious.
(Ex. 1008, pp 5=6).

139. While Li's system fits squarely in the detecting intrusions (it is directed toward improving IDSs), it clearly anticipates the need for post-detection response, such as coordinated research or notification to other parties on the network.

140. Piccard, which is in the same field of endeavor, is directed to the identification of malware and its sources:

Techniques are currently available to detect and remove malware. But as malware evolves, techniques for detecting and removing malware should also evolve. Accordingly, current techniques for detecting and removing malware are not always satisfactory and will likely not be satisfactory in the future. Current techniques for detecting and removing malware often use definitions of known malware to scan files of a protected computer. However, it is often difficult to initially locate malware in order to generate the definitions, particularly since malware can evolve. ***In particular, it would be desirable to identify sources of malware, such that definitions can be generated or updated to account for evolving malware.*** In addition, identification of sources of malware would allow a blacklist of Web sites to be generated.

(Ex. 1007, [003], emphasis added).

141. Piccard teaches a "malware detection module" as the first operation in its method, "to analyze a file to determine that the file includes potential malware (block 200)." (Ex. 1007, [0019]). Several possible techniques for such detection are disclosed, including monitoring for suspicious activity patterns:

In the illustrated embodiment, the malware detection module compares the file with a set of malware definitions to determine if the file includes potential malware. The set of malware definitions can include representations of malware, suspicious activities that are indicative of or that are common to malware, or both. For example, the set of malware definitions can include a hash value or a digital signature of malware. Such as one that is generated using Message Digest 5 ("MD5"). In this example, the malware detection module generates a hash value for the file and compares the hash value of the file with a set of hash values of malware to determine whether there is a sufficient match. As another example, the set of malware definitions can include a Cyclical Redundancy Code ("CRC") of a portion of malware. In this example, the malware detection module generates a CRC for the file and compares the CRC of the file with a set of

CRCs of malware to determine whether there is a sufficient match. ***As a further example, the set of malware definitions can include suspicious activities related to third-party cookies or related to entries or modifications of registry files of an operating system.*** (Ex. 1007, [0020], emphasis added).

142. A POSITA would have recognized that Li's teachings of detecting suspected malware through behavior monitoring align with Piccard's disclosures of its "malware detection module". Because Piccard discloses the "malware detection module" as a discrete component with multiple possible implementations but a defined output (i.e., the suspected file), a POSITA would have expected a high degree of success implementing Li's techniques as part of Piccard's malware detection module.

143. Furthermore, because Li anticipates the value of correlation with multiple types of activity information (*see* Li, pp. 3-4), a POSITA would have been motivated to include, as a further source of activity information, Piccard's disclosures of using the web history log to determine the source of the suspected malware.

144. Li also provides motivation for post-detection reaction to intrusions, for which a POSITA would have sought to use Piccard's "Web site identification module" and "reporting module" to better respond to suspected malware in order to prevent its recurrence. (*See* Li, p. 1: "The cost of the shut downs and cleaning up the infection was estimated to be around \$1 billion.").

145. Li is also clear that relevant information about malicious activity is available across the network, and that such information should be examined together and used to improve the security of the entire network: "Obtaining a system-wide view is very helpful for intrusion detection, as related malicious activities may appear in several hosts. Correlating this information can help detect attacks that otherwise may not be obvious." (Ex. 1008, pp. 5-6).

B. Claim 1 is rendered obvious by Li in view of Piccard

146. Each of the limitations of Claim 1 of the ‘244 patent is rendered obvious by Li in view of Piccard as described below.

1. Limitation 1[a]

147. Limitation 1[a] of the ‘244 patent reads:

A method for identifying an origin of suspected pestware activity on a computer, the method comprising:

148. I understand that the preamble of a patent claim is not always considered limiting. However, to the extent that the preamble of Claim 1 of the ‘244 patent is considered limiting, Li and Piccard both disclose the preamble of Claim 1.

149. Li describes its goal as follows: “This paper proposes a unified logging architecture, called UCLog, that effectively explores activity correlations for intrusion detection.” (Ex. 1008, p. 2). Li is particularly interested in identifying the origin of the intrusion: “At an intrusion, UCLog can be used to automatically identify the source and the execution path of the intrusion, whereas most current systems require an administrator to manually examine the logs to find such information.” (*Id.*).

150. Li describes “intrusions” and the state of the art of intrusion detection systems as follows:

Computer systems are vulnerable targets for terrorists and criminals to break-in. ***Intrusions can compromise the integrity, confidentiality, or availability of the systems and the information they hold*** [8]...

An intrusion detection system (IDS) attempts to detect attacks by monitoring system or network behavior. While many existing IDSs require manual definitions of normal and abnormal behavior (intrusion signatures) [37], recent work has shown that it is possible to identify abnormalities automatically using machine learning or data mining techniques [15, 5, 7, 18, 23] to generate models or rules, which the IDS can use to detect intrusions.

(Ex. 1008, p. 1, emphasis added).

151. Li's description of "intrusions" is similar to the broad definition in the '244 patent specification of "pestware", and a POSITA would have found them to refer to substantially the same field of endeavor.

Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as "malware," "spyware" or "pestware." ***These types of programs generally act to gather information about a person or organization—often without the person or organization's knowledge.*** Some pestware is highly malicious. Other pestware is non-malicious but may cause issues with privacy or system performance. And yet other pestware is actually beneficial or wanted by the user. Unless specified otherwise, "pestware" as used herein refers to any of these programs that collects information about a person or an organization. (Ex. 1001, 1:43-54).

152. Piccard also teaches a method for identifying the source of malware by examining web traffic history: "Embodiments of the invention include systems of managing malware. In one embodiment, a system includes a malware detection module configured to analyze a file of a protected computer to determine that the file is associated with malware. The system also includes a Web site identification module configured to search a download history log of the protected computer to identify a Web site from which the file was downloaded." (Ex. 1007, [0005]).

153. As I discussed in Section IX.A., a POSITA would have been motivated to combine Li with Piccard. Li is directed to the identification of pestware and recognizes the efficiency and accuracy gained through correlating multiple sources of activity information in that identification. Li further teaches the need to correlate information on a network level using, for example network logs, a concept related to Piccard's identification and reporting modules. Piccard teaches identifying the origin of pestware (e.g., a URL) and sharing that information with a research entity in order to provide better detection capabilities to nodes on the network. Furthermore, Piccard teaches a "malware detection module" with various exemplary techniques for detecting malware.

Li's techniques are directed to the same purpose as Piccard's "malware detection module", and a POSITA would have expected a high likelihood of success in combining those teachings.

154. Thus, Li and Piccard both disclose limitation 1[a].

2. Limitation 1[b]

155. Limitation 1[b] of the '244 patent reads:

monitoring, with a kernel-mode driver, activity on the computer;

156. Li discloses limitation 1[b] of the '244 patent.

157. Li teaches the use of "activity logging" to aggregate information from various sources on the system to better trace intrusions:

Activity logging is a mechanism to collect the information about past activities that the system or administrator can use to characterize access behavior, audit user accounts, replay operations, or detect anomalies and rollback to a previous state [16, 39]. For example, the system can log every user login/logout to monitor any possible break-ins. Research issues about logging include what and how to log, how to manage and protect logs, and how to use the logged data.

Event logging in Microsoft Windows provides a standard, centralized way for applications to record important software and hardware events [27]. By default, there are three types of logs: security, system, and application. (Ex. 1008, p. 2).

158. Li discloses multiple types of loggers that observe various aspects of system activity, including kernel APIs, network activity, file system events, user behavior, system events, and application events. Its descriptions of these general categories of loggers are as follows:

Kernel API Logger: A kernel API logger should record system calls, the parameters of those calls, and the time at which they are made.

Network Logger: A network logger could record all packets passing through the system, all packet headers received by the system, or even record higher, summary-level information (e.g. Cisco NetFlows [10]). The actual implementation would depend on the desired circumstances.

File-System Logger: A file-system logger might detect changes to significant blocks, such as the password file, or detect unusual access patterns. For example, a process which attempts to touch every block once could be harmless, such as a disk defragmentation process, or it could be some user attempting to illegally duplicate the disk.

User Behavior Logger: It would be useful to have some module that records a user's interactions with the system, such as when each user logs in and out.

System Events Logger: A facility that monitors the children, files, and directories created by some process would be useful in many situations.

Application-Specific Logger: Additional application-specific loggers can be added as needed.
(Ex. 1008, p. 5).

159. Li gives particular detail about the kernel API logger implementation: "System Call Log – Generated by APISpy32, contains a list of system calls executed by all running processes on the system. We extract the system call, process ID, and time of execution." (Ex. 1008, p. 9). A POSITA would have understood system calls to be API calls. (See Section V.B.). As discussed in Section V.C., A POSITA would further have understood that Li's use of APISpy32 involves monitoring such system API calls through a "kernel-mode driver"...

160. After Li's loggers collect these events, they transmit them to monitors, which do some processing and store the events in the activity database. In general terms:

Loggers collect information about various aspects of system behavior and send this data to monitors. Monitors watch [sic] system behavior and collaborate with each other through the centralized activity database. Monitors filter and compress audit data collected by loggers, and compare this data with existing models or signatures. The centralized activity database stores and manages audit data in an organized way.
(Ex. 1008, p. 4).

161. In more detail, Li explains the function of the logger, further illustrating how it collects data about activity on the system:

A logger records information about the behavior of a particular part of the environment and transmits that data to the corresponding monitor. For ease of deployment, a middleware layer of loggers with a common programming interface should be implemented. Thus, hiding the details of how to process each log from the other components. Each logging module must be coupled with the subsystem from which the logger is collecting audit data. For example, the network subsystem should have a logger module to collect network activities. Similarly, the user command shell also needs a logger to record user-entered commands.

(Ex. 1008, p. 5).

To reduce the performance overhead, loggers do very little processing but instead stream the data to the corresponding monitor.

(*Id.*, p. 6).

162. Thus, Li discloses limitation 1[b].

3. Limitation 1[c]

163. Limitation 1[c] of the '244 patent reads:

generating an activity log on a file storage device of the computer from the kernel-mode driver;

164. Li discloses limitation 1[c] of the '244 patent.

165. Li discloses generating an activity log in terms of the component logs used by its centralized system. These include logs for network events, system call events, process information events, and others.

We use several different loggers to capture data from multiple vantage points. For network behavior we use WinDump [14], a Windows port of the popular tcpdump. We use TDIMonitor version 1.01 [34], a logging utility that monitors the Transport Driver Interface. The information provided by WinDump and TDIMonitor are combined into the network traffic monitor and the activity database in order to associate a particular process ID with each network connection. To capture system call behavior we use APISpy32 v3.0 [21] with some modifications that allow us to capture the system time of each system call. Due to some limitations of APISpy32 we also use FileMon v5.01 [35] to capture certain file I/O operations. We use the built-in Windows 2000 security logging features in order to capture process information (start time, end time, process ID, and parent process ID) that we later use in correlating the results from different logs.

(Ex. 1008, p. 8).

166. These logs collect data about events. Li explains:

A monitor may also combine several log entries from different loggers. For example, if several different loggers generate information from the same event (for example, a file I/O system call is recorded by both the system call logger and the file I/O logger. In this case the monitor combines those logs to a unified format.

(Ex. 1008, p. 6).

167. At least one of the loggers is specifically described as being generated from kernel-level monitoring (which a POSITA would have known is from a kernel-mode driver). Li describes this as a “Kernel API Logger: A kernel API logger should record system calls, the parameters of those calls, and the time at which they are made.” (Ex. 1008, p. 5). A POSITA would have known that TDIMonitor is a kernel-level monitor as are file-system loggers. (See Section V.C.).

168. With respect to this limitation, Li also explains that the monitor is responsible for storing the collected (and, potentially, compressed or deduplicated) logs in the activity database. The database stores the data in a unified format and in a set of tables for the aggregated data. The aggregated database is also an activity log generated on a file storage device of the computer. Because at least some of its data is from kernel-mode logs (as discussed above), it is also generated “from” the kernel-mode driver:

When a monitor receives audit data from loggers, it needs to reduce the data size by pruning unnecessary information such as highly variable parameters, user variable names, and any other unpredictable information. To further pack the audit data, the monitor can compress the packet statistics to connection-level information. A monitor may also combine several log entries from different loggers. For example, if several different loggers generate information from the same event (for example, a file I/O system call is recorded by both the system call logger and the file I/O logger. In this case, the monitor combines those logs to a unified format. The data the monitors store in the activity database must be stored in a unified format, so that the rule learning and processing module can understand any entry.

(Ex. 1008, p. 6).

The activity database consists of a set of tables, each corresponding to the filtered log entries from a monitor.
(*Id.*).

169. A POSITA would have understood that the storage of the database is commonly on a file storage device of the computer.

170. Thus, Li discloses limitation 1[c].

4. Limitation 1[d]

171. Limitation 1[d] of the ‘244 patent reads:

receiving, from a user via an interface of the computer, a time of interest relating to a suspicion of pestware on the computer, wherein the time of interest includes a time interval;

172. Li discloses limitation 1[d] of the ‘244 patent.

173. Li discloses a user interface that is used to query the database and to generate correlations of the aggregated activity. Li explains that a user interface is a core component of an Intrusion Detection System (IDS).

Since log files contain information about past activities, the data collected by a logging system can be used by an IDS to detect signs of suspicious behavior.

A generic IDS contains three major components: sensors, analyzers, **and user interface [2]**. Sensors are responsible for collecting data, such as network packets and log files. The analyzer determines if an intrusion has occurred by analyzing the data from sensors. **The user interface enables an administrator to view output from the system and to control the behavior of the IDS.**

(Ex. 1008, p. 3, emphasis added).

174. The purpose of the interface is to make it easier for an administrator to query the system for answers to questions about behavior.

At an intrusion, **a system administrator who is assigned to analyze the intrusion might ask questions like “What process was running when a**

large spike of network activity occurred yesterday at midnight?" In most current systems, the administrator needs to examine the logs manually in order to find out the answers to his/her questions. But in UCLog, the activity database supports various queries to provide answers to these questions. To further simplify the administrator's job, some simple tools can be built on top of the activity database ***with a user-friendly interface and relieve the administrator from writing SQL statements.*** (Ex. 1008, p. 7, emphasis added).

175. As shown above, Li discloses receiving a time of interest related to suspicions of pestware as part of the user interface. To repeat the key words from the previous citation:

At an intrusion, a system administrator who is assigned to analyze the intrusion might ask questions like "What process was running when a large spike of ***network activity occurred yesterday at midnight?"*** (Ex. 1008, p. 7, emphasis added).

176. Li also discloses that times of interest involve time intervals: "Correlations between logs exist when two log entries have the same or related values for one or more fields. For example, a network connection might be associated with a given system call ***if the time is within a certain distance*** and both belong to the same process." (Ex. 1008, p. 7, emphasis added).

177. The system disclosed by Li is able to use rules that specifically work with time periods.

To model normal behavior, the rule generator retrieves the training data from the activity database. It is convenient to query particular data from the activity database to generate instance-specific rules via SQL statements. ***For example, to generate rules for normal system call sequences of a specific application in a specific time period,*** the generator only needs to query the data of normal system call logs with the WHERE conditions of processName and timeStamp of an SQL statement. (Ex. 1008, p. 7, emphasis added).

178. Thus, Li discloses limitation 1[d].

5. Limitation 1[e]

179. Limitation 1[e] of the '244 patent reads:

issuing a timestamp after receiving the time of interest

180. Li discloses limitation 1[e] of the '244 patent.

181. Li discloses that all system call event records in the aggregated log file have timestamps associated with them.

The activity database consists of a set of tables, each corresponding to the filtered log entries from a monitor. Each table describes some measurable and distinct aspect of system behavior. For example, in our prototype implementation, there is a table of system calls which records <processID, processName, systemCall, date, **timeStamp**, etc>, a table for network connections recording <startTime, duration, sourceHost, destHost, bytesFromSource, bytesFromDest, processId, etc>, and another for behavior recording <date, time, eventCode, pid, user, parentID>. (Ex. 1008, p. 6, emphasis added).

182. Li further discloses that the SQL statements issued to the system to obtain information use timestamps.

To model normal behavior, the rule generator retrieves the training data from the activity database. It is convenient to query particular data from the activity database to generate instance-specific rules via SQL statements. ***For example, to generate rules for normal system call sequences of a specific application in a specific time period***, the generator only needs to query the data of normal system call logs with the WHERE conditions of processName ***and timeStamp of an SQL statement***. (Ex. 1008, p. 7, emphasis added).

183. Although the above citation from Li only describes a specific type of query (e.g., retrieving training data from the activity database), a POSITA would have understood that all time-related SQL queries related to data with timestamps could include timestamps in the query. That is, a POSITA would have known that queries for data with a particular timestamp could include a timestamp in the query. Thus, a POSITA would have understood that the SQL queries issued for events with specific times could include a timestamp.

184. To the extent that a POSITA would not have understood this, a POSITA would have found it obvious to issue a timestamp as part of the SQL processes described above..

185. Thus, Li discloses limitation 1[e].

6. Limitation 1[f]

186. Limitation 1[f] of the '244 patent reads:

identifying, based upon the time of interest, indicia of pestware, wherein the identifying is initiated by the issuing the timestamp; and.

187. Li discloses limitation 1[f] of the '244 patent.

188. Li discloses identifying pestware such as viruses.

We added a file I/O monitor that watches the c:\recycler\ directory. ***The Yaha virus used in our experiments creates several files inside this directory.*** This is actually a common technique that current worms and viruses often use to conceal their whereabouts. Our new file I/O monitor watches any I/O into this directory and generate warnings upon file creations.

The Windows address book is often compromised by a virus and used to find other potential victims. ***The Yaha virus reads the address book and sends itself to the contacts.*** We decided that for testing purposes, no process should be able to secretly access the address book other than Outlook Express or Explorer. In fact, in recent versions of Outlook if you try and access the Contacts information from another program you are prompted for permission. So, our new monitor also looks for any access to the address book by an unauthorized process.
(Ex. 1008, p. 10).

189. Li also discloses that based on one suspicious event, the system will look for additional indications of intrusion (including malware or pestware such as the Yaha virus), by examining additional data within the same time period.

Figure 1(a) shows the results of the system call trace collected when the system is infected by the Yaha virus. ***As shown on the curve, there are some patches of increased abnormal behavior, but it is difficult to determine if any intrusion has really taken place, especially for the first abnormal spike.*** It is possible that either the user has just issued a command or tried a new feature that he has never tried before, or that a real intrusion occurs, i.e., the user just opened an e-mail attachment that compromises the local address book and fires off infected e-mail to other people in the address book. By examining the system call behavior alone, these types of questions can be difficult to answer because the system calls alone may not look all that abnormal.

When we examine the network traffic corresponding to the same time period, we can see an obvious increase in the network traffic as shown in Fig. 1(b). This means that the abnormal increase in the network traffic confirms the abnormal behavior in the system call sequence, therefore, the IDS can report an intrusion warning with a higher confidence. (Ex. 1008, p. 4, emphasis added).

190. As discussed with respect to limitation 1[e], a POSITA would have understood that searching within a time period would include issuing timestamps or, at the very least, it would have been obvious. Thus, the searching and identifying of the additional indications of pestware is initiated by issuing the timestamp.

191. Thus, Li discloses limitation 1[f].

7. Limitation 1[g]

192. Limitation 1[g] of the ‘244 patent reads:

accessing, using a hardware processor of the computer, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of an externally networked source that is suspected of originating pestware.

193. Li in view of Piccard discloses limitation 1[g] of the ‘244 patent.

194. Li discloses collecting data from various networking-related log files including a “Network Logger,” the “WinDump Log,” and the “TDIMonitor Log.”

Network Logger: A network logger could record all packets passing through the system, all packet headers received by the system, or even record higher, summary-level information (e.g. Cisco NetFlows [10]). The actual implementation would depend on the desired circumstances. (Ex. 1008, p. 6).

WinDump Log – Generated by WinDump. It contains the contents of the UDP and TCP packet headers of all traffic to and from the client machine.

(Ex. 1008, p. 10).

TDIMonitor Log – Generated by TDIMonitor. Captures network traffic at a different layer than WinDump. This logger is able to associate a process ID with each connection. This information is combined with the WinDump log to give us a process ID with each packet header.
(Ex. 1008, p. 10).

195. Each of these logs used by Li would include at least a portion of a recorded history of externally networked sources that the computer received files from. Moreover, Li discloses using this information in identifying pestware.

When we examine the network traffic corresponding to the same time period, we can see an obvious increase in the network traffic as shown in Fig. 1(b). This means that the abnormal increase in the network traffic confirms the abnormal behavior in the system call sequence, therefore, the IDS can report an intrusion warning with a higher confidence.
(Ex. 1008, p. 4, emphasis added).

196. Piccard also teaches using a download history log, which is at least a portion of a recorded history of externally networked sources, to identify the external source (e.g., URL) of a file suspected of being malware:

Embodiments of the invention include systems of managing malware. In one embodiment, a system includes a malware detection module configured to analyze a file of a protected computer to determine that the file is associated with malware. The system also includes a Web site identification module configured to search a download history log of the protected computer to identify a Web site from which the file was downloaded.
(Ex. 1007, [0005]).

197. In more detail, Piccard discloses that the “Web site identification module” is invoked after a determination of malware by the “malware detection module”. A POSITA would have found it obvious to have the “malware detection module” either be or be combined with Li’s IDS as described in limitation 1[f]. Li discloses identifying potential malware including the specific names of viruses. A POSITA would have understood a component that identifies malware to be a “malware detection module.” Thus, it would have been obvious to have Li’s IDS trigger

Piccard's "Web site identification module." Piccard indicates that the "Web site identification module" then accesses the history log to determine the external source:

In particular, once a file is downloaded using the Web browser 114, the malware detection module 118 analyzes the file to determine whether the file includes potential malware. ***If the file is determined to include potential malware, the Web site identification module 120 determines a Web address from which the file was received.*** In particular, the Web site identification module 120 ***accesses the history log 116 to identify the Web address of the file.*** In such manner, the Web site identification module 120 can identify a Web site from which the file was downloaded and, thus, can identify the Web site as a potential malware distribution site. (Ex. 1007, [0015]).

198. Piccard's "a Web address from which the file was received" teaches the claimed "so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of an externally networked source that is suspected of originating pestware".

199. As I discussed in Section IX.A., a POSITA would have been motivated to combine Li with Piccard. Li is directed to the identification of pestware and recognizes the efficiency and accuracy gained through correlating multiple sources of activity information in that identification. Li further teaches the need to correlate information on a network level using, for example network logs, a concept related to Piccard's identification and reporting modules. Similar to Li's Activity Log that, as described above in limitation 1[g], consists of a set of tables, Piccard discloses a "relational database" that is used to store data relating to identified malware. This database includes the distribution site: "As illustrated in FIG. 1, the database 126 also includes a list of malware distribution sites to alert a user about Web sites that are known to distribute malware or that are suspected of distributing malware... [and] can be implemented as, for example, a relational database in which information is organized using a set of tables." (Ex. 1007, [0014]). A POSITA would have recognized that both Li and Piccard's tables are compatible with each other and that Piccard's information would have served as another source for Li's correlation.

200. Piccard also teaches identifying the origin of pestware (e.g., a URL) and sharing that information with a research entity in order to provide better detection capabilities to nodes on the network. A POSITA would have found this an obvious improvement to Li's reporting system that identified abnormalities. As Li explains,

For normal traces, UCLog reports lower abnormality than the baseline system (see Fig. 4(b)). For some traces, the percentage abnormality with UCLog is significantly lower. For example, with n10, the baseline system reports 1.4% abnormality, whereas UCLog reports only 0.8% abnormality. Therefore, if 1% is the cutoff, the baseline would report a false intrusion but UCLog would not.
(Ex. 1008, p. 10).

201. A POSITA would have been motivated to extend reports of abnormality with information about potential malware distribution sites which would have further reduced false positives and reduced reports of "false intrusions".

202. Furthermore, Piccard teaches a "malware detection module" with various exemplary techniques for detecting malware. Li's techniques are directed to the same purpose as Piccard's "malware detection module", and a POSITA would have expected a high likelihood of success in combining those teachings.

203. Thus, Li in view of Piccard discloses limitation 1[g].

8. Limitation 1[h]

204. Limitation 1[h] of the '244 patent reads:

wherein the recorded history of externally networked sources is stored on the file storage device.

205. Li in view of Piccard discloses limitation 1[h] of the '244 patent.

206. Li discloses examining data in networking-related log files as discussed with respect to limitation 1[g] (e.g., networking log, WinDump log, and TDIMonitor log). A POSITA would have understood the logs of these systems to generally be stored on the local computer.

207. Even if the networking logs were not stored locally, all logs are aggregated into the database. The aggregated log is stored locally (e.g., in the central activity database):

Loggers collect information about various aspects of system behavior and send this data to monitors. Monitors watch [sic] system behavior and collaborate with each other through the centralized activity database. Monitors filter and compress audit data collected by loggers, and compare this data with existing models or signatures. ***The centralized activity database stores and manages audit data in an organized way.*** (Ex. 1008, p. 4, emphasis added).

208. Either way, Li discloses the recorded history of externally networked sources stored on the file storage device.

209. Piccard discloses using web browser download logs. A POSITA would have known that typically browsers store their download logs on the local file storage device. Piccard also discloses this explicitly:

As illustrated in FIG.1, ***the memory 112 also stores a history log 116***, which is maintained by the Web browser 114 to provide a record of browsing events. ***In particular, when a file is accessed and downloaded, the Web browser 114 records a Web address of the file in the history log 116.*** (Ex. 1007, [0013], emphasis added).

In particular, the Web site identification module 120 accesses the history log 116 to identify the Web address of the file. In such manner, the Web site identification module 120 can identify a Web site from which the file was downloaded and, thus, can identify the Web site as a potential malware distribution site. (Ex. 1007, [0015], emphasis added).

210. I have already explained why a POSITA would have been motivated to combine Li in view of Piccard in Section IX.A., and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

211. Thus, Li in view of Piccard discloses limitation 1[h].

212. Thus, I conclude that Li in view of Piccard renders Claim 1 obvious to a POSITA.

C. Claim 2 is rendered obvious by Li in view of Piccard

213. Claim 2 of the ‘244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Li in view of Piccard. Each of the limitations of Claim 2 of the ‘244 patent is rendered obvious by Li in view of Piccard as described below.

1. Limitation 2[a]

214. Limitation 2[a] of the ‘244 patent reads:

The method of claim 1, wherein the identifying includes: accessing at least one log of historical events on the computer

215. Li discloses the additional requirements of limitation 2[a].

216. As I explained with reference to limitation 1[f], Li teaches examining historical event data from logs (e.g., system call activity and network activity):

Figure 1(a) shows the results of the system call trace collected when the system is infected by the Yaha virus. ***As shown on the curve, there are some patches of increased abnormal behavior, but it is difficult to determine if any intrusion has really taken place, especially for the first abnormal spike.*** It is possible that either the user has just issued a command or tried a new feature that he has never tried before, or that a real intrusion occurs, i.e., the user just opened an e-mail attachment that compromises the local address book and fires off infected e-mail to other people in the address book. By examining the system call behavior alone, these types of questions can be difficult to answer because the system calls alone may not look all that abnormal.

When we examine the network traffic corresponding to the same time period, we can see an obvious increase in the network traffic as shown in Fig. 1(b). This means that the abnormal increase in the network traffic confirms the abnormal behavior in the system call sequence, therefore, the IDS can report an intrusion warning with a higher confidence. (Ex. 1008, p. 4, emphasis added).

217. Furthermore, Li teaches that activity from the various log sources are aggregated in an “activity database” to aid in querying for related information:

The activity database stores and manages audit data in a very organized way. It is implemented as an efficient relational database management system in order to present monitors with an interface to query the database to find correlated activities or to trace to the root and the path of an intrusion. (Ex. 1008, p. 6).

218. Similar to Li’s Activity Log that, as described above in limitation 1[g], consists of a set of tables, Piccard discloses a “relational database” that is used to store data relating to identified malware. This database includes the distribution site: “As illustrated in FIG. 1, the database 126 also includes a list of malware distribution sites to alert a user about Web sites that are known to distribute malware or that are suspected of distributing malware... [and] can be implemented as, for example, a relational database in which information is organized using a set of tables.” (Ex. 1007, [0014]). A POSITA would have recognized that both Li and Piccard’s tables are compatible with each other and that Piccard’s information would have served as another source for Li’s correlation

219. Thus, Li in view of Piccard discloses limitation 2[a].

2. Limitation 2[b]

220. Limitation 2[b] of the ‘244 patent reads:

locating at least one event in the at least one log that corresponds to the time of interest; and

221. Li discloses the additional requirements of limitation 2[b].

222. Li discloses locating events that correspond to the time of interest by identifying events that are within a time interval of the time of interest. Li discloses that the SQL queries can specify a time of interest in the user interface (e.g., time windows around a specific event as

discussed below). For example, Li discloses identifying events in the network traffic logs that corresponds to the time of interest.

When we examine the network traffic corresponding to the same time period, we can see an obvious increase in the network traffic as shown in Fig. 1(b). This means that the abnormal increase in the network traffic confirms the abnormal behavior in the system call sequence, therefore, the IDS can report an intrusion warning with a higher confidence. (Ex. 1008, p. 4, emphasis added).

223. Thus, Li in view of Piccard discloses limitation 2[b].

3. Limitation 2[c]

224. Limitation 2[c] of the '244 patent reads:

determining that the at least one event is associated with the indicia of pestware.

225. Li discloses the additional requirements of limitation 2[c].

226. As discussed with respect to limitation 2[b], Li discloses examining network traffic events corresponding to a specific time period. Li also discloses determining that these events are associated with indicia of pestware by identifying them as having abnormal increases in the network traffic, confirming that an intrusion of a virus (pestware) has occurred.

Figure 1(a) shows the results of the system call trace collected when the system is **infected by the Yaha virus**. As shown on the curve, there are some patches of increased abnormal behavior, but it is difficult to determine if any intrusion has really taken place, especially for the first abnormal spike. It is possible that either the user has just issued a command or tried a new feature that he has never tried before, or that a real intrusion occurs, i.e., the user just opened an e-mail attachment that compromises the local address book and fires off infected e-mail to other people in the address book. By examining the system call behavior alone, these types of questions can be difficult to answer because the system calls alone may not look all that abnormal.

When we examine the network traffic corresponding to the same time period, we can see an obvious increase in the network traffic as shown in

Fig. 1(b). This means that the abnormal increase in the network traffic confirms the abnormal behavior in the system call sequence, therefore, the IDS can report an intrusion warning with a higher confidence.

(Ex. 1008, p. 4, emphasis added).

227. Thus, Li in view of Piccard discloses limitation 2[c].

228. Thus, I conclude that Li in view of Piccard renders Claim 2 obvious to a POSITA.

D. Claim 3 is rendered obvious by Li in view of Piccard

229. Claim 3 of the '244 patent, which depends from Claim 2, is rendered obvious to a POSITA by Li in view of Piccard. Claim 3 reads,

The method of claim 2, wherein the determining that the at least one event is associated with the indicia of pestware includes following references in the at least one log to a suspected pestware object.

230. Li discloses the additional requirements of Claim 3.

231. Li discloses identifying a suspected malware file and/or process, which is a pestware object, based on data within the log files. For example, Li discloses identifying all program names that belong to a given parent (program) as shown in Table 1, reproduced hereinbelow (annotated):

Table 1. Micro-benchmark results

ID	Query	Avg. Time
1	Select all system calls occurring in some time range	0.196 sec
2	Select all system calls belonging to some process in a given trace	0.425 sec
3	Select all program names belonging to some parent	0.011 sec
4	Select all process IDs receiving packets in some time range	0.082 sec
5	Query 4 and then Query 1 (with processID filtering)	0.6 sec

232. Li also discloses looking up the process name (which is either identical to or related to the file name)

For example, to generate rules for normal system call sequences **of a specific application** in a specific time period, the generator only needs to query the data of normal system call logs **with the WHERE conditions of processName** and timeStamp of an SQL statement.

(Ex. 1008, p. 7, emphasis added).

233. The system is explicitly designed to enable the tracing of intrusions to specific processes. Li explains, the system “...also supports queries from a monitor to find out correlated activities watched by other monitors, **or to trace to the specific processes** or users that directly or indirectly cause the occurring intrusion.” (Ex. 1008, p. 4, emphasis added).

234. Thus, I conclude that Li in view of Piccard renders Claim 3 obvious to a POSITA.

E. Claim 4 is rendered obvious by Li in view of Piccard

235. Claim 4 of the ‘244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Li in view of Piccard. Claim 4 reads,

The method of claim 3, wherein following the references includes following at least one reference in the activity log from the event to a suspected pestware process.

236. Li in view of Piccard discloses the additional requirements of Claim 4.

237. As I explained with respect to Claim 3, Li discloses identifying file objects and running processes associated with the suspected pestware. A POSITA would have known that a “process” is an executable file that is running. For example, Li discloses identifying all program names that belong to a given parent (program) as shown in Table 1, reproduced hereinbelow (annotated):

Table 1. Micro-benchmark results

ID	Query	Avg. Time
1	Select all system calls occurring in some time range	0.196 sec
2	Select all system calls belonging to some process in a given trace	0.425 sec
3	Select all program names belonging to some parent	0.011 sec
4	Select all process IDs receiving packets in some time range	0.082 sec
5	Query 4 and then Query 1 (with processID filtering)	0.6 sec

238. Li also discloses looking up the process name (which is either identical to or related to the file name)

For example, to generate rules for normal system call sequences *of a specific application* in a specific time period, the generator only needs to query the data of normal system call logs *with the WHERE conditions of processName* and timeStamp of an SQL statement.
(Ex. 1008, p. 7, emphasis added).

239. The system is explicitly designed to enable the tracing of intrusions to specific processes. Li explains, the system "...also supports queries from a monitor to find out correlated activities watched by other monitors, *or to trace to the specific processes* or users that directly or indirectly cause the occurring intrusion." (Ex. 1008, p. 4, emphasis added).

240. Piccard teaches analyzing the file identified by a malware detection device including whether or not the file acts suspiciously. Piccard explains:

As a further example, the set of malware definitions can include suspicious activities related to third-party cookies or related to entries or modifications of registry files of an operating system.
(Ex. 1007, [0020]).

241. A POSITA would have understood that only a file that is executing (i.e., is a process) is capable of suspicious activities such as modification to the registry file of an operating system. In combination with Li, the log data that identifies the file is also combined with the activities of the associated process.

242. As I discussed in Section X.A., a POSITA would have been motivated to combine Li and Piccard. I incorporate my analysis herein by reference.

243. Thus, I conclude that Li in view of Piccard renders Claim 4 obvious to a POSITA.

F. Claim 5 is rendered obvious by Li in view of Piccard

244. Claim 5 of the '244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Li in view of Piccard. Claim 5 reads,

The method of claim 2, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

245. Li discloses the additional requirements of Claim 5.

246. I have already explained with respect to limitation 1[c] that the generated component logs and the generated aggregated log, are activity logs. As also explained with respect to limitation 1[c], these logs are also event logs.

247. Li also discloses that the component logs also include file information and monitoring software. For example, Li references FileMon: “[Log] Created by FileMon. It records all of the file I/O on the system. This information is combined with the system call log and stored in the database” (Ex. 1008, p. 9).

248. Thus, I conclude that Li in view of Piccard renders Claim 5 obvious to a POSITA.

G. Claim 6 is rendered obvious by Li in view of Piccard

249. Claim 6 of the ‘244 patent, which depends from Claim 1, is rendered obvious by Li in view of Piccard. Claim 6 reads:

The method of claim 1, wherein the user input is in response to an alert provided to the user regarding a detected suspicious activity on the computer.

250. Li in view of Piccard discloses the additional requirements of Claim 6.

251. Li discloses that abnormal behavior can generate alerts. For example, “Then if the file I/O monitor detects many file accesses from an ordinary user [during nighttime hours], it can

generate a warning that an intrusion is likely taking place.” (Ex. 1008, p. 6). Piccard also discloses an alert mechanism when malware is suspected:

In the illustrated embodiment, once the Web site identification module identifies the Web site, a reporting module (e.g., the reporting module 122) reports information regarding the Web site to a remotely-located computer that is connected to the protected computer. This information can identify the Web site as a potential malware distribution site, such as in terms of the domain name of the Web site. Alternatively, or in conjunction, this information can identify the file as including potential malware, Such as in terms of the name of the file or the URL of the file. It is also contemplated that this information can include a represent tation of the file or can identify suspicious activities related to the file.

In the illustrated embodiment, the reporting module also alerts a user of the protected computer about the Web site.

(Ex. 1007, [0023]-[0024]).

252. Moreover, Li discloses additional investigation when abnormal behavior is detected (e.g., when an alert indicates a potential issue). For example, Li identifies noting abnormal syscalls when the Yaha virus infects a system, but the abnormality is not sufficient to suspect an intrusion. Instead, this triggers additional investigation:

Figure 1(a) shows the results of the system call trace collected when the system is infected by the Yaha virus. ***As shown on the curve, there are some patches of increased abnormal behavior, but it is difficult to determine if any intrusion has really taken place, especially for the first abnormal spike.*** It is possible that either the user has just issued a command or tried a new feature that he has never tried before, or that a real intrusion occurs, i.e., the user just opened an e-mail attachment that compromises the local address book and fires off infected e-mail to other people in the address book. By examining the system call behavior alone, these types of questions can be difficult to answer because the system calls alone may not look all that abnormal.

When we examine the network traffic corresponding to the same time period, we can see an obvious increase in the network traffic as shown in Fig. 1(b). This means that the abnormal increase in the network traffic confirms the abnormal behavior in the system call sequence, therefore, the IDS can report an intrusion warning with a higher confidence. (Ex. 1008, p. 4, emphasis added).

253. Li discloses that a system with a user interface for inputting questions related to an investigation. Li explains that a user interface is a core component of an Intrusion Detection System (IDS).

Since log files contain information about past activities, the data collected by a logging system can be used by an IDS to detect signs of suspicious behavior.

A generic IDS contains three major components: sensors, analyzers, **and user interface [2]**. Sensors are responsible for collecting data, such as network packets and log files. The analyzer determines if an intrusion has occurred by analyzing the data from sensors. **The user interface enables an administrator to view output from the system and to control the behavior of the IDS.**

(Ex. 1008, p. 3, emphasis added).

254. The purpose of the interface is to make it easier for an administrator to query the system for answers to questions about behavior.

At an intrusion, **a system administrator who is assigned to analyze the intrusion might ask questions like "What process was running when a large spike of network activity occurred yesterday at midnight?"** In most current systems, the administrator needs to examine the logs manually in order to find out the answers to his/her questions. But in UCLog, the activity database supports various queries to provide answers to these questions. To further simplify the administrator's job, some simple tools can be built on top of the activity database **with a user-friendly interface and relieve the administrator from writing SQL statements.**

(Ex. 1008, p. 7, emphasis added).

255. As shown above, Li discloses receiving a time of interest related to suspicions of pestware as part of the user interface. To repeat the key words from the previous citation:

At an intrusion, a system administrator who is assigned to analyze the intrusion might ask questions like "What process was running when a large spike of **network activity occurred yesterday at midnight?**"

(Ex. 1008, p. 7, emphasis added).

256. A POSITA would have understood that the investigation performed in response to the abnormal syscalls caused by the Yaha virus would result in user inputting additional queries

into the system as described above. To the extent a POSITA would not understand this it would have been obvious.

257. Thus, I conclude that Li in view of Piccard renders Claim 6 obvious to a POSITA.

H. Claim 7 is rendered obvious by Li in view of Piccard

258. Claim 7 of the ‘244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Li in view of Piccard. Claim 7 reads,

The method of claim 1, including: reporting the identity of the externally networked source to an externally networked pestware research entity so as to enable the pestware research entity to research whether the externally networked source is a source of pestware.

259. Li in view of Piccard discloses this Claim.

260. As discussed with respect to limitation 1[h], Piccard discloses identifying the identity of the externally networked source. Piccard also discloses reporting this information to a remotely-located computer. Piccard explains:

The third operation illustrated in FIG. 2 is to report that the Web site is a potential malware distribution site (block 204). In the illustrated embodiment, once the Web site identification module identifies the Web site, a reporting module (e.g., the reporting module 122) reports information regarding the Web site **to a remotely-located computer** that is connected to the protected computer. This information can identify the Web site as a potential malware distribution site, such as in terms of the domain name of the Web site... This information as well as any additional relevant **information can be analyzed at the remotely-located computer to determine whether the file does, in fact, include malware and whether the Web site is, in fact, a malware distribution site.**

(Ex. 1007, [0023], emphasis added).

261. A POSITA would have understood that a “remotely-located computer” that “analyze[s]... to determine whether ... the Web site is, in fact, a malware distribution site” is an externally networked pestware research entity. This “remotely-located computer” serves the

purpose of analyzing the information to determine whether the website is indeed a source of malware and subsequently reporting its findings. The “reporting module” is responsible for transmitting the necessary information to do such analysis.

262. I have already explained why a POSITA would be motivated to combine Li and Piccard in Section IX.A. and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

263. Thus, I conclude that Li in view of Piccard renders Claim 7 obvious to a POSITA.

I. Claim 8 is rendered obvious by Li in view of Piccard

264. Claim 8 of the ‘244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Li in view of Piccard. Claim 8 reads,

The method of claim 1, wherein the externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

265. Li in view of Piccard discloses Claim 8.

266. As explained with respect to limitation 1[g], Piccard discloses identifying the external network source of the potential pestware. Piccard discloses identifying the source by both IP address and URL.

As illustrated in FIG.1, the memory 112 also stores a history log 116, which is maintained by the Web browser 114 to provide a record of browsing events. In particular, when a file is accessed and downloaded, the Web browser 114 records a Web address of the file in the history log 116. A Web address typically specifies a location of a file within a Web site. ***For example, a Web address can be a Uniform Resource Identifier (“URI) of a file, such as a Uniform Resource Locator (“URL) of the file. It is also contemplated that a Web address can be defined in various other ways, such as using an Internet Protocol (“IP) address or any other identifier of a source of a file.***

(Ex. 1007, [0013], emphasis added).

267. I have already explained why a POSITA would be motivated to combine Li and Piccard in Section IX.A. and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

268. Thus, I conclude that Li in view of Piccard renders Claim 8 obvious to a POSITA.

J. Claim 9 is rendered obvious by Li in view of Piccard

269. Each of the limitations of Claim 9 of the ‘244 patent is rendered obvious by Li in view of Piccard as described below.

1. Limitation 9[a]

270. Limitation 9[a] of the ‘244 patent reads:

A system for identifying a source of suspected pestware on a computer, the system comprising:

271. I understand that the preamble of a patent claim is not always considered limiting. However, to the extent that the preamble of Claim 9 of the ‘244 patent is considered limiting, Li and Piccard both disclose the preamble of Claim 9.

272. Li describes a system for dealing with intrusions. “This paper proposes a unified logging architecture, called UCLog, that effectively explores activity correlations for intrusion detection.” (Ex. 1008, p. 2). Li is particularly interested in identifying the origin of the intrusion: “At an intrusion, UCLog can be used to automatically identify the source and the execution path of the intrusion, whereas most current systems require an administrator to manually examine the logs to find such information.” (*Id.*).

273. Li describes “intrusions” and the state of the art of intrusion detection systems as follows:

Computer systems are vulnerable targets for terrorists and criminals to break-in. ***Intrusions can compromise the integrity, confidentiality, or availability of the systems and the information they hold*** [8]...

An intrusion detection system (IDS) attempts to detect attacks by monitoring system or network behavior. While many existing IDSs require manual definitions of normal and abnormal behavior (intrusion signatures) [37], recent work has shown that it is possible to identify abnormalities automatically using machine learning or data mining techniques [15, 5, 7, 18, 23] to generate models or rules, which the IDS can use to detect intrusions.
(Ex. 1008, p. 1, emphasis added).

274. Li's description of "intrusions" is similar to the broad definition in the '244 patent specification of "pestware", and a POSITA would have found them to refer to substantially the same field of endeavor.

Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as "malware," "spyware" or "pestware." ***These types of programs generally act to gather information about a person or organization—often without the person or organization's knowledge.*** Some pestware is highly malicious. Other pestware is non-malicious but may cause issues with privacy or system performance. And yet other pestware is actually beneficial or wanted by the user. Unless specified otherwise, "pestware" as used herein refers to any of these programs that collects information about a person or an organization.
(Ex. 1001, 1:43-54).

275. Piccard also teaches a system for identifying the source of malware by examining web traffic history: "Embodiments of the invention include systems of managing malware. In one embodiment, a system includes a malware detection module configured to analyze a file of a protected computer to determine that the file is associated with malware. The system also includes a Web site identification module configured to search a download history log of the protected computer to identify a Web site from which the file was downloaded." (Ex. 1007, [0005]).

276. As I discussed in Section IX.A., a POSITA would have been motivated to combine Li with Piccard. Li is directed to the identification of pestware and recognizes the efficiency and accuracy gained through correlating multiple sources of activity information in that identification. Li further teaches the need to correlate information on a network level using, for example network

logs, a concept related to Piccard's identification and reporting modules. Piccard teaches identifying the origin of pestware (e.g., a URL) and sharing that information with a research entity in order to provide better detection capabilities to nodes on the network. Furthermore, Piccard teaches a "malware detection module" with various exemplary techniques for detecting malware. Li's techniques are directed to the same purpose as Piccard's "malware detection module", and a POSITA would have expected a high likelihood of success in combining those teachings.

277. Thus, both Li and Piccard disclose limitation 9[a].

2. Limitation 9[b]

278. Limitation 9[b] of the '244 patent reads:

a hardware processor:

279. Both Li and Piccard disclose limitation 9[b].

280. Li discloses an implementation on a computer system: "We have implemented the UCLog architecture on a real system running Microsoft Outlook Express on Windows 2000." (Ex. 1008, p. 2). A POSITA would have understood a computer system that runs Windows 2000 to have a hardware processor.

281. Piccard also discloses a processor (CPU). "In the illustrated embodiment, the protected computer 102 is a client computer and includes conventional client computer components, including a Central Processing Unit ("CPU") 108 that is connected to a network connection device 110 and a memory 112." (Ex. 1007, [0012]).

282. Thus, both Li and Piccard disclose limitation 9[b].

3. Limitation 9[c]

283. Limitation 9[c] of the '244 patent reads:

a file storage device coupled to the hardware processor:

284. Both Li and Piccard disclose limitation 9[c].

285. Li discloses generating an activity log in terms of the component logs used by its centralized system. These include logs for network events, system call events, process information events, and others.

We use several different loggers to capture data from multiple vantage points. For network behavior we use WinDump [14], a Windows port of the popular tcpdump. We use TDIMonitor version 1.01 [34], a logging utility that monitors the Transport Driver Interface. The information provided by WinDump and TDIMonitor are combined into the network traffic monitor and the activity database in order to associate a particular process ID with each network connection. To capture system call behavior we use APISpy32 v3.0 [21] with some modifications that allow us to capture the system time of each system call. Due to some limitations of APISpy32 we also use FileMon v5.01 [35] to capture certain file I/O operations. We use the built-in Windows 2000 security logging features in order to capture process information (start time, end time, process ID, and parent process ID) that we later use in correlating the results from different logs. (Ex. 1008, p. 8).

286. These logs collect data about events. Li explains:

A monitor may also combine several log entries from different loggers. For example, if several different loggers generate information from the same event (for example, a file I/O system call is recorded by both the system call logger and the file I/O logger. In this case the monitor combines those logs to a unified format. (Ex. 1008, p. 6).

287. Li discloses that the logged data is stored in an activity database that consists of a set of tables. “The activity database consists of a set of tables, each corresponding to the filtered log entries from a monitor.” (Ex. 1008, p. 6). This is also illustrated in Li’s Figure 2, reproduced hereinbelow:

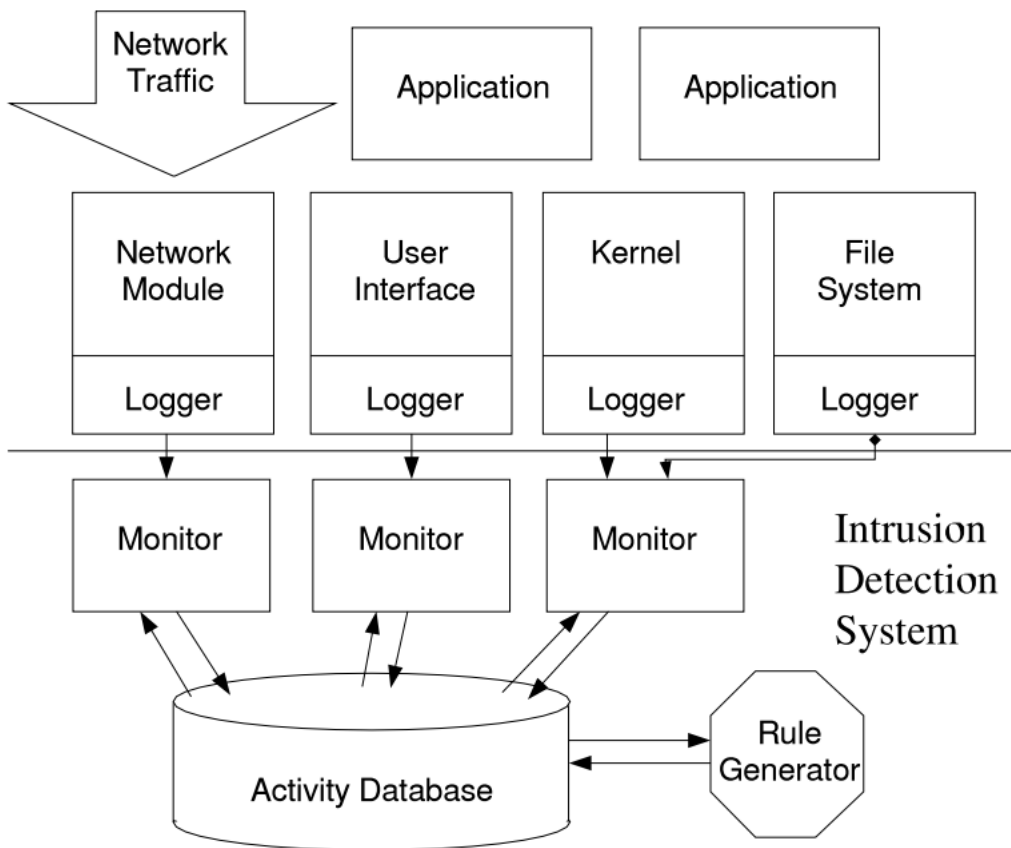


Figure 2. UCLog architecture

288. A POSITA would have understood that the activity database, including its tables, are stored on a file storage device connected to the hardware processor..

289. Piccard also discloses a file storage device that is coupled to the hardware processor:

An embodiment of the invention relates to a computer program product with a computer-readable medium including computer code or executable instructions thereon for performing a set of computer-implemented operations. The medium and computer code can be those specially designed and constructed for the purposes of the invention, or they can be of the kind well known and available to those having ordinary skill in the computer Software arts. ***Examples of computer-readable media include: magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as Compact Disc-Read Only Memories ("CD-ROMs) and holographic devices; magneto-opticalmedia Such as floptical disks; and***

hardware devices that are specially configured to store and execute computer code, such as Application-Specific Integrated Circuits (ASICs), Programmable Logic Devices (“PLDs”), Read Only Memory (“ROM”) devices, and Random Access Memory (“RAM”) devices. (Ex. 1007, [0026], emphasis added).

290. Piccard discloses that data is stored in the tables of a relational database. “As illustrated in FIG. 1, the database 126 also includes a list of malware distribution sites to alert a user about Web sites that are known to distribute malware or that are suspected of distributing malware... [and] can be implemented as, for example, a relational database in which information is organized using a set of tables.” (Ex. 1007, [0014]).

291. Thus, both Li and Piccard disclose limitation 9[c].

4. Limitation 9[d]

292. Limitation 9[d] of the ‘244 patent reads:

a timestamp module configured to be executed by the hardware processor, to generate a time of interest including a time interval in response to detected activity on the computer that is indicative of pestware, to issue a timestamp after generating the time of interest, and to initiate, in response to the issuance of the timestamp, identification of indicia of pestware:

293. Li discloses limitation 9[d].

294. The language of limitation 9[d] is almost identical to the language of limitations 1[d], 1[e], and 1[f]. The only significant difference is that limitation 9[d] requires a timestamp “module” to perform the operations. I incorporate my analysis of limitations 1[d], 1[e], and 1[f] into this section by reference.

295. Li also discloses a timestamp module. Li discloses a database table for system calls that records the timestamp of each event.

The activity database consists of a set of tables, each corresponding to the filtered log entries from a monitor. Each table describes some measurable and distinct aspect of system behavior. For example, in our prototype implementation, there is a table of system calls which records <processID, processName, systemCall, date, **timeStamp**, etc>, a table for network connections recording <startTime, duration, sourceHost, destHost, bytesFromSource, bytesFromDest, processId, etc>, and another for behavior recording <date, time, eventCode, pid, user, parentID>. (Ex. 1008, p. 6, emphasis added).

296. Li further discloses that the SQL statements issued to the system to obtain information use timestamps.

To model normal behavior, the rule generator retrieves the training data from the activity database. It is convenient to query particular data from the activity database to generate instance-specific rules via SQL statements. ***For example, to generate rules for normal system call sequences of a specific application in a specific time period***, the generator only needs to query the data of normal system call logs with the WHERE conditions of processName ***and timeStamp of an SQL statement***. (Ex. 1008, p. 7, emphasis added).

297. A POSITA would have understood that the database table for system calls and the rule generator that creates rules for finding events within a time period are collectively a timestamp module required by the claim.

298. Thus, Li discloses limitation 9[d].

5. Limitation 9[e]

299. Limitation 9[e] of the '244 patent reads:

a research portion configured to be executed by the hardware processor and to access at least one log on a file storage device of the computer to relate the time of interest to at least one externally networked source of potential pestware activity on the computer;;

300. Li in view of Piccard discloses limitation 9[e].

301. The language of limitation 9[e] is similar to the language of Claim 2. The only significant difference is “a research portion” that performs the operations.

302. A POSITA would have understood the IDS components, as disclosed by Li, that access a log file combined with the Web site identification module that identifies the external networked source to be a “research portion.”

303. I have already explained why a POSITA would be motivated to combine Li and Piccard in Section IX.A. and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

304. Thus, Li in view of Piccard discloses limitation 9[e].

6. Limitation 9[f]

305. Limitation 9[f] of the ‘244 patent reads:

a kernel-mode driver configured to monitor activity on the computer and to generate an activity log that is included within the at least one log; and:

306. Li discloses limitation 9[f].

307. The language of limitation 9[f] is similar to the language of limitation 1[b]. There are no substantive changes. I incorporate my analysis of limitation 1[b] into this section by reference.

308. Thus, Li discloses limitation 9[f].

7. Limitation 9[g]

309. Limitation 9[g] of the ‘244 patent reads:

a reporting portion configured to be executed by the hardware processor and to generate a report that identifies the at least one externally networked source of potential pestware activity:

310. Li in view of Piccard discloses limitation 9[g].

311. The language of limitation 9[g] is similar to limitation 1[g] with respect to identifying the at least one externally networked source of potential pestware activity. The only significant change is that the operations must be performed by “a reporting portion” and that the reporting portion must “generate a report.” I incorporate my analysis of limitation 1[g] by reference.

312. As discussed with reference to limitation 1[g], Li discloses a reporting system that identified abnormalities. As Li explains,

For normal traces, UCLog reports lower abnormality than the baseline system (see Fig. 4(b)). For some traces, the percentage abnormality with UCLog is significantly lower. For example, with n10, the baseline system reports 1.4% abnormality, whereas UCLog reports only 0.8% abnormality. Therefore, if 1% is the cutoff, the baseline would report a false intrusion but UCLog would not.
(Ex. 1008, p. 10).

313. Piccard also discloses generating a report including information about potential malware distribution sites.

The third operation illustrated in FIG. 2 is to report that the Web site is a potential malware distribution site (block 204). In the illustrated embodiment, once the Web site identification module identifies the Web site, ***a reporting module (e.g., the reporting module 122) reports information regarding the Web site to a remotely-located computer that is connected to the protected computer.*** This information can identify the Web site as a potential malware distribution site, such as in terms of the domain name of the Web site. Alternatively, or in conjunction, this information can identify the file as including potential malware, such as in terms of the name of the file or the URL of the file. It is also contemplated that this information can include a representation of the file or can identify suspicious activities related to the file. ***This information as well as any additional relevant information can be analyzed at the remotely-located computer to determine whether the file does, in fact, include malware and whether the Web site is, in fact, a malware distribution site.*** If the Web site is determined to be a malware distribution site, a new or updated set of malware definitions can be generated based on content within the Web site,

and the new or updated set of malware definitions can be provided to the protected computer. In addition, content within the Web site can be monitored on a periodic or other basis for new or updated malware. Also, a new or updated list of malware distribution sites can be generated so as to identify the Web site, and the new or updated list of malware distribution sites can be provided to the protected computer.
(Ex. 1007, [0023], emphasis added).

314. A POSITA would have understood the reporting module to be the “reporting portion” and that reporting information by the reporting portion is “generating a report.” A POSITA would have been motivated to extend reports of abnormality with information about potential malware distribution sites which would have further reduced false positives and reduced reports of “false intrusions”.

315. Thus, Li in view of Piccard discloses limitation 1[g].

316. Thus, I conclude that Li in view of Piccard renders Claim 9 obvious to a POSITA.

K. Claim 10 is rendered obvious by Li in view of Piccard

317. Claim 10 of the ‘244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Li in view of Piccard. Claim 10 reads,

The system of claim 9, wherein the at least one log includes information that relates the time of interest to an event, and wherein the at least one log includes information that relates the event to a suspected pestware object, and wherein the at least one log includes information that relates the suspected pestware object to the at least one externally networked source of potential pestware activity.

318. Li in view of Piccard discloses Claim 10.

319. The language of Claim 10 is similar to language found in Claims 2, 4, and 8. I incorporate my analysis of those three claims into this section by reference.

320. Thus, I conclude that Li in view of Piccard renders Claim 10 obvious to a POSITA.

L. Claim 11 is rendered obvious by Li in view of Piccard

321. Claim 11 of the '244 patent, which depends from Claim 10, is rendered obvious to a POSITA by Li in view of Piccard. Claim 11 reads,

The system of claim 10, including: a heuristics module configured to be executed by the hardware processor and to evaluate whether the event is a suspected pestware-related event before the research portion accesses information in the at least one log that relates the event to the suspected pestware object.

322. Li discloses the additional requirements of Claim 11.

323. Li discloses using RIPPER, which it describes as a "data mining" technique, to train on recognizing good and bad sequences of system calls. The rules generated from this component are used to identify abnormal system calls that indicate a possible intrusion.

The next step is to train the detection system by applying a datamining algorithm, RIPPER [12, 13], to the trace data to extract useful behavior patterns. The rules generated by RIPPER are used to classify normal and abnormal system call sequences.

After obtaining the rules from the training data, the next step is prediction and post-processing of test data. Similar to Lee's postprocessing approach in [23], each sequence of system calls in the testing trace is predicted as "\normal" or "\abnormal" according to the rules. The predictions are then divided into regions of size $3l + 1$. If the number of abnormal prediction in a region is greater than a half, we predict the region as "\abnormal". Next, we take an overlapping sliding window of size 500 and pass it over the sequence of predictions. For each system call sequence, we calculate the number of abnormal predictions in this sliding window. Results are graphed in Fig. 1 and give us an idea of the behavior of a given abnormal trace.

Figure 1(a) shows the results of the system call trace collected when the system is infected by the Yaha virus. As shown on the curve, there are some patches of increased abnormal behavior, but it is difficult to determine if any intrusion has really taken place, especially for the first abnormal spike. It is possible that either the user has just issued a command or tried a new feature that he has never tried before, or that a real intrusion occurs, i.e., the user just opened an e-mail attachment that compromises the

local address book and fires off infected e-mail to other people in the address book. By examining the system call behavior alone, these types of questions can be difficult to answer because the system calls alone may not look all that abnormal.

(Ex. 1008, pp. 3-4, emphasis added).

324. As indicated by Li, the heuristics of the RIPPER system suggest a possible intrusion. Li explains in detail that the RIPPER system is used as part of the rule generation system. While one part of the system uses rules for known attacks, the RIPPER component is part of behavioral/heuristic detection that can identify previously unseen intrusions:

The behavior models are used to detect any anomalies, while the signatures are used to detect known intrusions [4].

To model normal behavior, the rule generator retrieves the training data from the activity database. It is convenient to query particular data from the activity database to generate instance-specific rules via SQL statements. For example, to generate rules for normal system call sequences of a specific application in a specific time period, the generator only needs to query the data of normal system call logs with the WHERE conditions of processName and timeStamp of an SQL statement. Then the training data is formatted and fed to modeling algorithms. In our current implementation, we use a data-mining algorithm, RIPPER, to generate rules for system call sequences, and statistics methods for other logs.

(Ex. 1008, p. 7).

325. This leads to further investigation using the research module, as discussed in limitation 9[e] in order to further ascertain whether an intrusion has, in fact, taken place.

326.

When we examine the network traffic corresponding to the same time period, we can see an obvious increase in the network traffic as shown in Fig. 1(b). This means that the abnormal increase in the network traffic confirms the abnormal behavior in the system call sequence, therefore, the IDS can report an intrusion warning with a higher confidence.

(Ex. 1008, p. 4, emphasis added).

327. Li also discloses other “statistics” techniques for data other log files.

In our current implementation, we use a data-mining algorithm, RIPPER, to generate rules for system call sequences, and statistics methods for other logs.
(Ex. 1008, p. 7).

328. As discussed in Section V.D., a POSITA would have understood that identifying abnormal system call sequences against training data (or other statistical models) is an example of a “heuristic” that makes an estimate or guess as to the classification of a test input. As I discussed in Section V.D., a “heuristics-based detection engine” may look for behaviors commonly exhibited by malware in order to identify suspected malware, even when other detection mechanisms would not have discovered it (e.g., a never-before-seen virus that does not yet have a signature in the database of an antivirus scanner). (See Section V.D.; see also Skoudis, pp. 11-12). Moreover, as disclosed in limitation 9[e], the access of the research portion is to identify the external network source. As explained within that limitation, the identification of the external network source is performed after malware detection.

329. Thus, I conclude that Li in view of Piccard renders Claim 11 obvious to a POSITA.

M. Claim 12 is rendered obvious by Li in view of Piccard

330. Claim 12 of the ‘244 patent, which depends from claim 11, is rendered obvious to a POSITA by Li in view of Piccard. Claim 12 reads,

The system of claim 11, wherein the heuristics module determines that the at least one event is suspected to be pestware-related by analyzing other events in connection with the event, wherein the other events are also related to the time of interest

331. Li discloses the additional requirements of Claim 12.

332. As explained with respect to Claim 11, Li discloses correlating multiple activity log events together, such as system calls and network traffic. Li explains that correlating data from

multiple event logs is a key design of the system: “[c]orrelating activities from all involved components and then analyzing them together would give the IDS a better picture of the system, increasing the detection accuracy and reducing the false positives.” (Ex. 1008, p. 1). The architecture disclosed by Li performs correlation: “[t]his paper proposes a unified logging architecture, called UCLog, that effectively explores activity correlations for intrusion detection.” (Ex. 1008, p. 2). Li states this again, stating “UCLog effectively exploits activity correlations between different components to improve the accuracy of intrusion detection and to reduce the number of false positives.” (*Id.*). Li also discloses that practical tests were conducted that proved the effectiveness: “we have studied how correlating activities from the system call sequence, network activity, and file I/O operations can improve the intrusion detection accuracy.” (*Id.*).

333. The activity database disclosed by Li is used to find correlations:

The activity database stores and manages audit data in a very organized way. It is implemented as an efficient relational database management system in order to present monitors with an interface to query the database to find correlated activities or to trace to the root and the path of an intrusion. (Ex. 1008, p. 6).

334. Li also uses information from the Windows 2000 Security Log in order to enable correlations between logs.

We use the built-in Windows 2000 security logging features in order to capture process information (start time, end time, process ID, and parent process ID) that we later use in correlating the results from different logs. (Ex. 1008, p. 8).

335. Li also states that using correlated data improved the ability to detect intrusions by a significant factor:

Notice that as we correlate more logs, our ability to detect intrusions increases. For example, in a30, the abnormality is 48% with correlations, whereas the abnormality without correlations only gives 19%, a factor of improvement of 2.5. (Ex. 1008, p. 10).

336. Moreover, Li explains in detail that the RIPPER system is used as part of the rule generation system. While one part of the system uses rules for known attacks, the RIPPER component is part of behavioral/heuristic detection that can identify previously unseen intrusions:

The behavior models are used to detect any anomalies, while the signatures are used to detect known intrusions [4].

To model normal behavior, the rule generator retrieves the training data from the activity database. It is convenient to query particular data from the activity database to generate instance-specific rules via SQL statements. For example, to generate rules for normal system call sequences of a specific application in a specific time period, the generator only needs to query the data of normal system call logs with the WHERE conditions of processName and timeStamp of an SQL statement. Then the training data is formatted and fed to modeling algorithms. In our current implementation, we use a data-mining algorithm, RIPPER, to generate rules for system call sequences, and statistics methods for other logs.
(Ex. 1008, p. 7).

337. Thus, I conclude that Li in view of Piccard renders Claim 12 obvious to a POSITA.

N. Claim 13 is rendered obvious by Li in view of Piccard

338. Claim 13 of the '244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Li in view of Piccard. Claim 13 reads,

The system of claim 9, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

339. This claim is virtually identical to Claim 5. I incorporate my analysis of Claim 5 herein by reference and conclude that Li in view of Piccard renders Claim 13 obvious to a POSITA.

O. Claim 14 is rendered obvious by Li in view of Piccard

340. Claim 14 of the '244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Li in view of Piccard. Claim 14 reads,

The system of claim 9, wherein the timestamp module is further configured to generate the time of interest as a point in time.

341. Li discloses the additional requirements of Claim 14.

342. I explained with respect to limitation 9[d] that the time of interest includes a time interval. However, Li also discloses a “point in time” as well. Specifically, Li discloses identifying a “point in time”. Li explains:

At an intrusion, a system administrator who is assigned to analyze the intrusion might ask questions like “What process was running when a large spike of **network activity occurred yesterday at midnight?**” (Ex. 1008, p. 7, emphasis added).

343. Thus, I conclude that Li in view of Piccard renders Claim 14 obvious to a POSITA.

P. Claim 15 is rendered obvious by Li in view of Piccard

344. Claim 15 of the '244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Li in view of Piccard. Claim 15 reads,

The system of claim 9, wherein the at least one externally networked source of potential pestware activity is identified by an identifier selected from the group consisting of an I.P. address and a URL.

345. This claim is virtually identical to Claim 8. I incorporate my analysis of Claim 8 herein by reference and conclude that Li in view of Piccard renders Claim 15 obvious to a POSITA.

Q. Claim 16 is rendered obvious by Li in view of Piccard

346. Claim 16 of the '244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Li in view of Piccard. Claim 16 reads,

The system of claim 9, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

347. This claim is **exactly** identical to Claim 13. I incorporate my analysis of Claim 13 herein by reference and conclude that Li in view of Piccard renders Claim 16 obvious to a POSITA.

X. GROUND 2: CLAIMS 1-16 ARE RENDERED OBVIOUS BY CHEN IN VIEW OF PICCARD

348. As I describe in greater detail hereinbelow, a POSITA would have found Claims 1-16 rendered obvious by Chen in view of Piccard.

A. Motivation to Combine Chen and Piccard

349. A POSITA would have been motivated to combine the teachings of Chen and Piccard. As I discussed in Section VIII.A., Chen is directed toward consolidating and correlating information from various log sources:

The focus of the research described in this paper is on the nature of the event information provided in commonly available computer and other logs and the extent to which it is possible to correlate such event information despite its heterogeneous nature and origins. The strategic purpose of the research has been to develop a means by which a consolidated repository of such information can be constituted and then queried in order to provide an investigator with post hoc event correlation for forensics purposes (ECF). The paper provides an account of the log processing techniques utilized, and the nature of the database and query engine that have been developed in our current prototype and two examples of scenarios investigated and identified by the current prototype.
(Ex. 1006, p. 1).

350. Chen discloses capability of ingesting logs in the several formats (“Apache Server Log”, “Windows 2000 Security Log”, “Door Log”, “Browser Cache Logs”, and “*NIX SYSLOG

– POP3, SSH, sendmail, smap”). (Ex. 1006, pp. 4-5). Chen also anticipates additional types of logs being used in its system: “It is expected that future parsers will likewise be regular expression based and we are reworking some of the other parsers to use the same approach. This will facilitate extensibility with regard to accommodating new logs or re-formatted versions of existing logs.” (Ex. 1006, p. 50).

351. Chen also discloses that the logs used in the forensic analysis can grow. In fact, the system disclosed by Chen was designed to incorporate new logs, new correlation of data, and otherwise expand as new information becomes available to the forensic investigator.

An overriding objective in the research and development has been that of extensibility: it must be relatively straightforward to incorporate new types of event and transactional logs and to accommodate log changes and configurable log formats that may vary from system to system and organization to organization. The ECF system must provide an extensible framework for the analysis and correlation of heterogeneous event and transaction records.
(Ex. 1006, p. 2).

The results presented in Section 4 above indicate that cross-correlation of event information emanating from heterogeneous logs potentially provides a powerful basis for the forensic identification of scenarios of interest even though only a very small base of common denominator information is being taken into account. Clearly, the inclusion of additional information from various log sources can only expand the potential usefulness of the technique.
(Ex. 1006, p. 9).

352. Piccard teaches steps to take after identification of malware downloaded from an external source:

Techniques are currently available to detect and remove malware. But as malware evolves, techniques for detecting and removing malware should also evolve. Accordingly, current techniques for detecting and removing malware are not always satisfactory and will likely not be satisfactory in the future. Current techniques for detecting and removing malware often use definitions of known malware to scan files of a protected computer. However, it is often difficult to initially locate malware in order to generate the definitions, particularly since malware can evolve. *In particular, it*

would be desirable to identify sources of malware, such that definitions can be generated or updated to account for evolving malware. In addition, identification of sources of malware would allow a blacklist of Web sites to be generated.

(Ex. 1007, [003], emphasis added).

353. Piccard teaches a “malware detection module” as the first operation in its method, “to analyze a file to determine that the file includes potential malware (block 200).” (Ex. 1007, [0019]). Several possible techniques for such detection are disclosed, including monitoring for suspicious activity patterns:

In the illustrated embodiment, the malware detection module compares the file with a set of malware definitions to determine if the file includes potential malware. The set of malware definitions can include representations of malware, suspicious activities that are indicative of or that are common to malware, or both. For example, the set of malware definitions can include a hash value or a digital signature of malware. Such as one that is generated using Message Digest 5 (“MD5”). In this example, the malware detection module generates a hash value for the file and compares the hash value of the file with a set of hash values of malware to determine whether there is a sufficient match. As another example, the set of malware definitions can include a Cyclical Redundancy Code (“CRC”) of a portion of malware. In this example, the malware detection module generates a CRC for the file and compares the CRC of the file with a set of CRCs of malware to determine whether there is a sufficient match. *As a further example, the set of malware definitions can include suspicious activities related to third-party cookies or related to entries or modifications of registry files of an operating system.*

(Ex. 1007, [0020], emphasis added).

354. A POSITA would have recognized that Chen’s teachings of forensically identifying malicious software, including software downloaded via browser, align with Piccard’s disclosures of its “malware detection module”. Because Piccard discloses the “malware detection module” as a discrete component with multiple possible implementations but a defined output (i.e., the suspected file), a POSITA would have expected a high degree of success implementing Chen’s techniques as part of Piccard’s malware detection module.

355. Furthermore, because Chen anticipates the ongoing expansion of multiple types of activity information as it becomes available (*see* Chen, pp. 2), a POSITA would have been motivated to include, as a further source of activity information, Piccard’s disclosures of using the web history log to determine the source of the suspected malware. Chen already discloses using browser logs and browser caches so, to the extent these do not include a web history log, it would have been obvious to add it.

356. Chen is also directed to forensic analysis. As Chen states, “The strategic purpose of the research has been to develop a means by which a consolidated repository of such event information can be constituted and then queried in order to provide an investigator with post hoc event correlation for forensics purposes (ECF).” (Ex. 1006, p. 1). Chen further teaches the need to correlate information on a network level using, for example, Apache logs. This concept is analogous in form and similar in purpose to concepts taught by Piccard’s identification and reporting modules. Piccard teaches identifying the origin of pestware (e.g., a URL) and sharing that information with a research entity in order to provide better detection capabilities to nodes on the network. Furthermore, Piccard teaches a “malware detection module” with various exemplary techniques for detecting malware. A POSITA would have been motivated to use Piccard especially if Apache log files are not available.

B. Claim 1 is rendered obvious by Chen in view of Piccard

357. Each of the limitations of Claim 1 of the ‘244 patent is rendered obvious by Chen in view of Piccard as described below.

1. Limitation 1[a]

358. Limitation 1[a] of the ‘244 patent reads:

A method for identifying an origin of suspected pestware activity on a computer, the method comprising:

359. I understand that the preamble of a patent claim is not always considered limiting. However, to the extent that the preamble of Claim 1 of the ‘244 patent is considered limiting, Chen and Piccard both disclose the preamble of Claim 1.

360. Chen discloses a method of enabling and improving forensic analysis of computer systems and software:

The strategic purpose of the research has been to develop a means by which a consolidated repository of such event information can be constituted and then queried in order to provide an investigator with post hoc event correlation for forensics purposes (ECF).
(Ex. 1006, p. 1).

361. Moreover, Chen explains that the purpose of the ECF is to identify malware, such as an exploit that attacks the Windows operating system: “... scenarios described in this section have been identified to demonstrate the capabilities of the ECF software... [such as the] [d]ownload and execution of an exploit allowing elevated privileges on a Windows 2000 workstation.” (Ex. 1006, p. 6). As discussed in Section V.E., a POSITA would have understood that “elevat[ing] privileges” is a means of installing malware software deep inside the Windows operating system in order to spy on the system without detection. Thus, Chen discloses identifying suspected pestware activity.

362. Moreover, Chen discloses identifying an origin of the suspected activity. For example, Chen discloses tracing the source of pestware back to the download URL:

The browser logs located on workstation W could be used to gain additional information about the different sites that the user visited, however we have not included those logs in this particular scenario as the browser cache did not contain a copy of the exploit file (due to restrictions on the type/size of files stored in the cache by browsers, as described above).
(Ex. 1006, p. 8).

363. See also, e.g., Figure 1 reproduced hereinbelow:

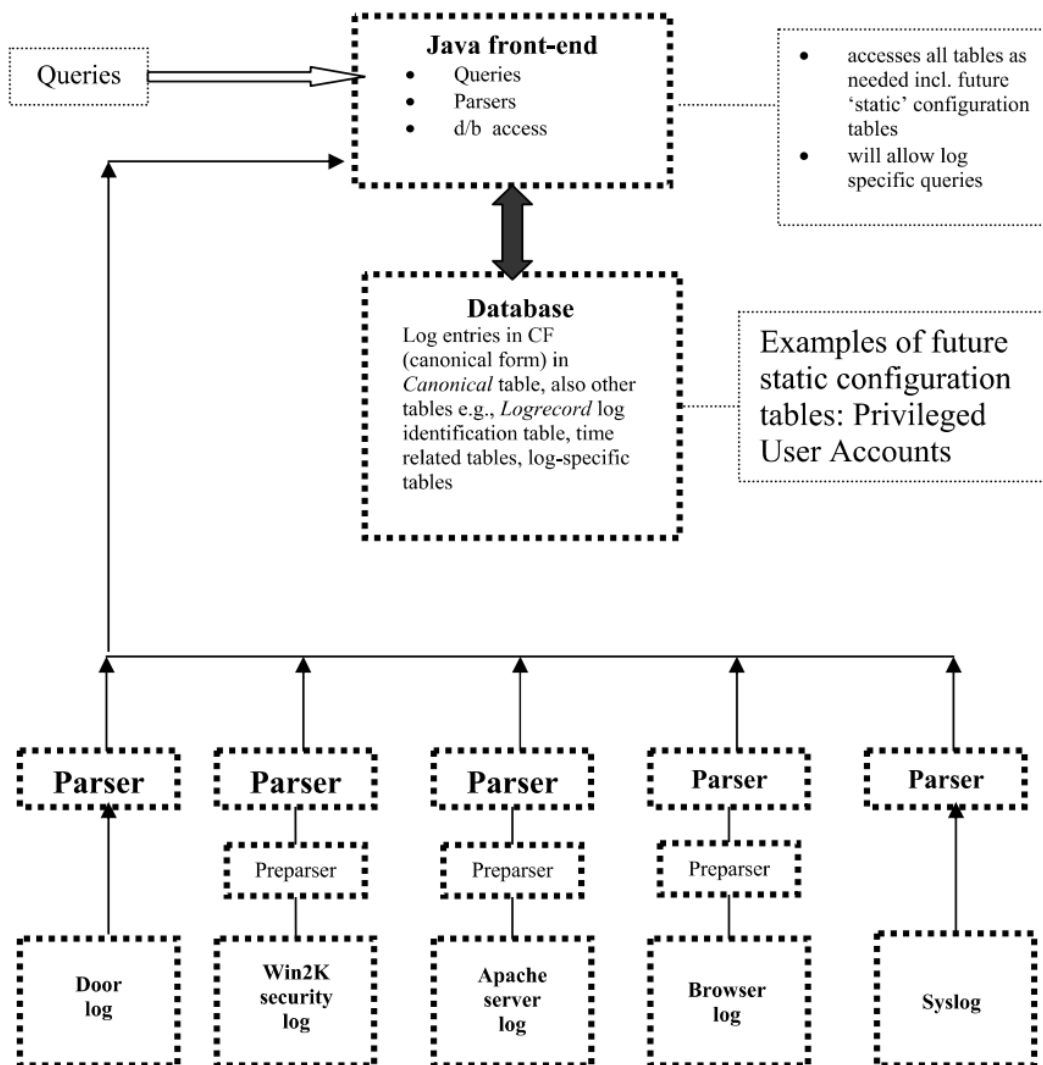


Figure 1: The ECF Architecture

364. Piccard also teaches a method for identifying the source of malware by examining web traffic history: “Embodiments of the invention include systems of managing malware. In one embodiment, a system includes a malware detection module configured to analyze a file of a protected computer to determine that the file is associated with malware. The system also includes a Web site identification module configured to search a download history log of the protected computer to identify a Web site from which the file was downloaded.” (Ex. 1007, [0005]).

365. As I discussed in Section X.A., a POSITA would have been motivated to combine Chen and Piccard. Chen is directed to the identification of processes exhibiting abnormal behavior and anticipates the need for appropriate responses to such processes. Piccard teaches identifying the origin of pestware that has been detected by techniques like those taught in Chen. Chen's techniques are directed to the same purpose as Piccard's "malware detection module", and a POSITA would have expected a high likelihood of success in combining those teachings.

366. Thus, Chen in view of Piccard discloses limitation 1[a].

2. Limitation 1[b]

367. Limitation 1[b] of the '244 patent reads:

monitoring, with a kernel-mode driver, activity on the computer;

368. Chen discloses limitation 1[b] of the '244 patent.

369. Chen discloses obtaining data from the "Windows 2000 Security Log". (See, e.g., many examples, Chen, pp. 4-5). This log is generated by a kernel-level component of the OS..

370. A POSITA would have understood that audit information found in the Windows 2000 Security Log is generated by the Security Reference Monitor. The Security Reference Monitor is a kernel component that performs the monitoring of security events.

Security Reference Monitor (SRM)— The component of the Windows 2000 Executive (kernel) that actually performs the security access checks, adjusts privileges, and generates audit messages when necessary. (Schmidt, Microsoft Windows 2000 Security Handbook, chapter 3).

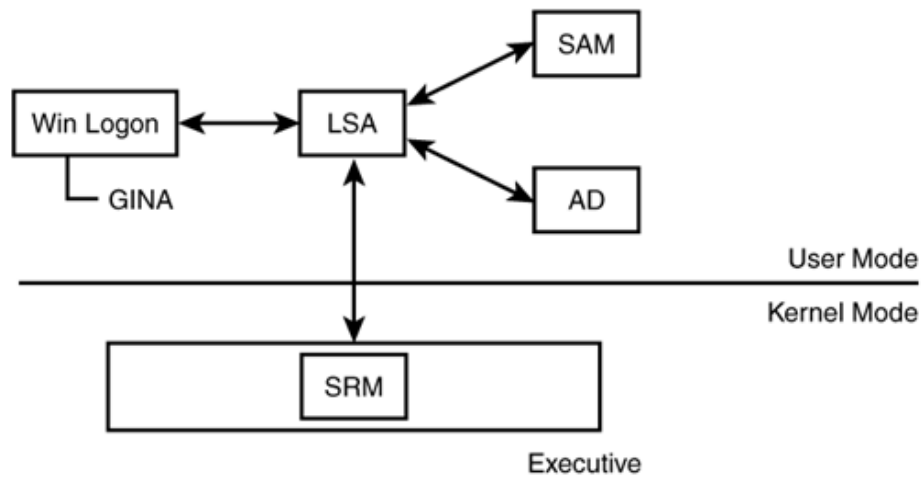
371. The Security Log entries in the Windows 2000 Security Log are recorded by Local Security Authority (LSA) based on alerts raised by the SRM:

The Local Security Authority (LSA) is a protected subsystem that maintains the information about all aspects of local security on a system. ... In general, the LSA performs the following functions: ... Manages the Audit policy and settings. When an audit alert is generated by the Security Reference

Monitor, the LSA is charged with writing that alert to the appropriate system log.”
(MSDN).

372. The following diagram illustrates the interaction of these components. As shown in the illustration, the SRM is a kernel-level module:

Figure 3.9. Windows 2000 security components.



373. A POSITA would have understood that, generally, security events are generated by the kernel and kernel-level modules. For example, kernels perform operations known as “traps” that are used to collect information during kernel scheduling of application execution. At least some security events include information trapped by the kernel.

In Figure 24.13 you can see the Event Viewer with the Security log file selected. Entries in the pane on the right-hand side represent events that were trapped by Windows 2000 based on security auditing.
(Schimdt, chapter 24)

374. I have also reproduced the figure 24.13, mentioned above, in the space below.

[illegible]

3. Limitation 1[c]

generating an activity log on a file storage device of the computer from the kernel-mode driver;

378. Chen discloses that systems produce many log files that Chen subsequently validates. For example, it includes logs of “system event[s], audit [events], and door [events].” NITA would have understood these types of events to be “activity” and the corresponding logs to be both event logs and activity logs.

The research described in this paper focuses on the nature of the event information provided in commonly available computer and other logs (**such as system event logs, audit logs, door logs** etc) and the extent to which it is possible to correlate such event information despite its heterogeneous nature and origins. The strategic purpose of the research has been to develop a means by which a consolidated repository of such event information can be constituted and then queried in order to provide an investigator with post hoc event correlation for forensics purposes (ECF). (Ex. 1006, p. 1, emphasis added).

379. Each of these logs is an activity log generated on a file storage device. As discussed with respect to limitation 1[b], at least the Windows 2000 Security Log is “from the kernel mode driver.” I incorporate my analysis of the Windows 2000 Security Log in limitation 1[b] into this section by reference.

380. Moreover, Chen discloses generating a new composite log combined from all the log sources. In order to do this, log records are standardized with information about the time of the event, the party that triggered the action, the action itself, and the receiver of the action.

The strategic purpose of the research has been to develop a means by which a consolidated repository of such event information can be constituted and then queried in order to provide an investigator with post hoc event correlation for forensics purposes (ECF).

The decision was made to opt for the former viz., for a canonical representation of what is essentially the minimal or base set of fields which appear in all event records viz., Time, Subject, Action, and Object. These fields map intuitively to almost all event records no matter how varied the events or systems on which they occur, and so a single database table (the Canonical table) which represents that information is consequently the central information store used for event correlation in ECF.” (Ex. 1006, pp. 1-2).

381. Because the information in the composite log includes data from the Windows 2000 Security Log, it is also from “the kernel module.”

382. See also, e.g., Figure 1 reproduced herein below:

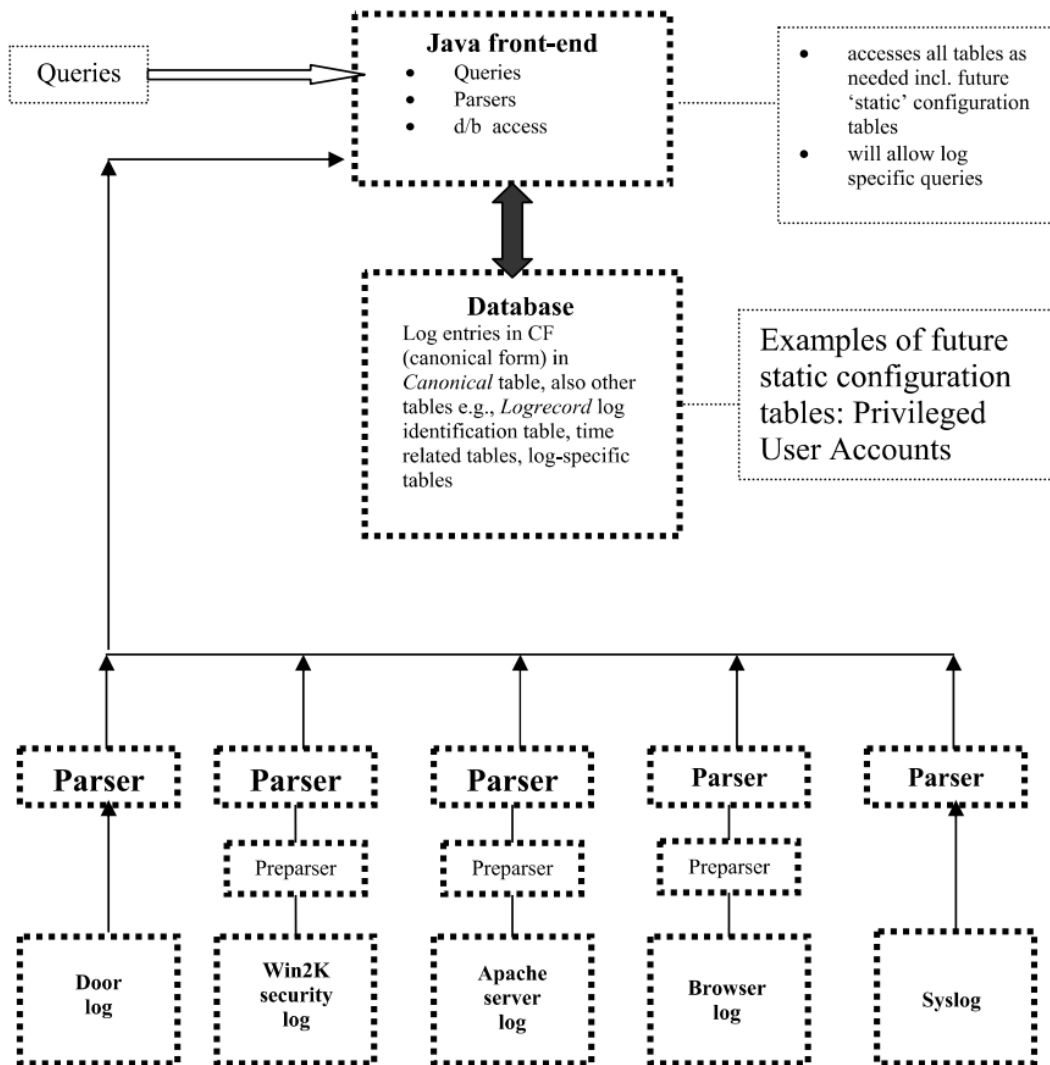


Figure 1: The ECF Architecture

383. Thus, Chen discloses limitation 1[c].

4. Limitation 1[d]

384. Limitation 1[d] of the '244 patent reads:

receiving, from a user via an interface of the computer, a time of interest relating to a suspicion of pestware on the computer, wherein the time of interest includes a time interval;

385. Chen discloses limitation 1[d] of the '244 patent.

386. Chen discloses a user interface, permitting SQL queries, for an administrator to query events. Chen discloses that the SQL queries can specify a time of interest in the user interface (e.g., time windows around a specific event as discussed below):

The investigator would presumably be interested in knowing times when the administrator account on the workstation was logged in to. ***The ECF tool Dynamic Query interface was used to generate the following query which will return a list of records relating to logins to the 'administrator' account on the workstation in question ('dsto')...***

This query returned a single record from our test logs which we do not show here due to space limitations. ***However, the record returned occurred at the time '2003-08-08 04:56:5' (UCT).*** The investigator would also be interested in other accounts that had been used to login to the workstation at around the same time. Therefore, in a similar fashion to the previous scenario ***we perform a query to determine all the users who logged onto the workstation 'dsto' in the time window commencing ten minutes prior to the identified records,*** and finishing ten minutes after it. The following SQL query, which was generated using the Dynamic Query interface, extracts the desired records:

```
SELECT      *      FROM      CANONICAL,      LOGRECORD      WHERE
CANONICAL.LOGID      =      LOGRECORD.LOGID      AND
CANONICAL.TIME      >=      '2003-08-08      04:46:51'      AND
CANONICAL.TIME      <=      '2003-08-08      05:06:51'      AND
CANONICAL.ACTION = 'Logon' AND LOGRECORD.LOGNODE =
'dsto';
```

(Ex. 1006, p. 8, emphasis added).

387. Chen's indication of a time of interest is a time interval as discussed above (e.g., the time windows commencing ten minutes prior to the identified records). Moreover, time and time querying in general are a core part of the system:

Time

Parsing of the time field of a log and insertion of the corresponding time value into the database takes place in three phases... The stringified representation of this is inserted into the Time field of the Canonical table while the binary representation is inserted into one of the two event-time related tables as starttime and stoptime. That table is keyed by Eventid and lists the event times of all events in the Canonical table. All querying is based on the starttime value.

(Ex. 1006, p. 4).

388. Additionally, Chen discloses a “Dynamic Query interface” as an interface to the system. The Dynamic Query interface permits users to select the data from the database graphically and using constraints.

Dynamic queries are satisfied largely by access to the main database table, the Canonical table with processing of time values relying upon the field starttime in one of the two event-time related tables...

In presenting a query, the operator has the option of selecting a subset of the Canonical fields to be displayed in the results by setting the appropriate checkboxes.

The query is built by selecting various constraints and relating them with AND and/or OR relationships. For example, if both Time and Subject are constraints, using the AND relationship means both Time and Subject constraints need to be satisfied, using the OR relationship means that the result set should include events that satisfy either constraint.

When a constraint is selected, the application populates the related drop down menu with all corresponding values currently in the database and this may then be used to simply choose the desired value as the constraint. The constraints can also be set to match a sub-string. For example, if SubjectType is set to ip_address and Subject is set to “contains 131.181” then all IP addresses in the 131.181.0.0 subnet will be matched and only records satisfying that constraint will be displayed (subject to other constraints also).

(Ex. 1006, p. 4).

389. A POSITA would have understood that the timing constraints identified by Chen above (e.g., “within 10 mintues”) are constraints that could be expressed by the constraints of the Dynamic Query interface.

390. In addition to the Dynamic Query interface, Chen also discloses a “Custom Query” interface, which is used to create more flexible SQL queries.

The Custom Query tab accepts an SQL query from the operator and displays the results in a table. It provides a means to execute more flexible queries than can be built using the Dynamic Query tab option.

(Ex. 1006, p. 5).

391. See also Figure 2, which illustrates part of the user interface for displaying the output of these queries:

entryid	rawtime	subject	subjecttype	object	objecttype	action	result	lognode	enter	exclu
9735	2003-08-08 03:48:00	KEVIN CHEN...	username	GP. S BLOCK RM 9826A	door	open	success	qut security	FALSE	FALSE
9771	2003-08-08 03:51:32	ISRC chen	username	528 ISRC/BND # Logon ID: (0x0,0x23F41F)	GENERIC...	Logon	unknown	bnd	FALSE	FALSE
9878	2003-08-08 03:51:53	chen	username	needham/needham	computer	ssh - login	success	bnd	FALSE	FALSE
10830	2003-08-08 03:52:30	chen	username	http://cdn.netscape.com/_media/wpsite/pop...	url	get	success	needham	FALSE	FALSE
10835	2003-08-08 03:52:32	chen	username	http://cdn.netscape.com/wpton_03/aolforbro...	url	get	success	needham	FALSE	FALSE
10846	2003-08-08 03:52:33	chen	username	http://cdn.netscape.com/wpt_tonr_03/20030...	url	get	success	needham	FALSE	FALSE
10847	2003-08-08 03:52:37	chen	username	http://cdn.netscape.com/wpton_03/dot_d2c9...	url	get	success	needham	FALSE	FALSE
10836	2003-08-08 03:52:37	chen	username	http://cdn.netscape.com/wpton_03/gradient	url	get	success	needham	FALSE	FALSE
10831	2003-08-08 03:52:40	chen	username	http://home.netscape.com/_ads/adsPopup2...	url	get	success	needham	FALSE	FALSE
10832	2003-08-08 03:53:45	chen	username	http://www.microsoft.com/library/include/ctre...	url	get	success	needham	FALSE	FALSE
10837	2003-08-08 03:53:45	chen	username	http://www.microsoft.com/library/toolbar/ima...	url	get	success	needham	FALSE	FALSE
10848	2003-08-08 03:54:18	chen	username	http://www-int.isrc.qut.edu.au/images/bg2.jpg	url	get	success	needham	FALSE	FALSE
10849	2003-08-08 03:54:36	chen	username	http://www-int.isrc.qut.edu.au/dev/ac/porn.html	url	get	success	needham	FALSE	FALSE

Figure 2: Results of query searching for records whose Subject contains the substring 'chen'.

392. Thus, Chen discloses limitation 1[d].

5. Limitation 1[e]

393. Limitation 1[e] of the '244 patent reads:

issuing a timestamp after receiving the time of interest

394. Chen discloses limitation 1[e] of the '244 patent.

395. Chen discloses issuing a timestamp after receiving the time of interest. For example, Chen discloses identifying a time of interest with respect to events of a specific time and the beginning and ending of a ten-minute window of the designated event. Chen discloses issuing timestamps related to these events in order for the system to find all events within the time period.

This query returned a single record from our test logs which we do not show here due to space limitations. **However, the record returned occurred at the time '2003-08-08 04:56:5' (UCT).** The investigator would also be interested in other accounts that had been used to login to the workstation at around the same time. Therefore, in a similar fashion to the previous scenario **we perform a query to determine all the users who logged onto the workstation 'dsto' in the time window commencing ten minutes prior to the identified records**, and finishing ten minutes after it. The following SQL query, which was generated using the Dynamic Query interface, extracts the desired records:

(Ex. 1006, p. 8, emphasis added).

Dynamic queries are satisfied largely by access to the main database table, the Canonical table with processing of time values relying upon the field starttime in one of the two event-time related tables...

When a constraint is selected, the application populates the related drop down menu with all corresponding values currently in the database and this may then be used to ***simply choose the desired value as the constraint***. The constraints can also be set to match a sub-string. For example, if SubjectType is set to ip_address and Subject is set to “contains 131.181” then all IP addresses in the 131.181.0.0 subnet will be matched and only records satisfying that constraint will be displayed (subject to other constraints also).

(Ex. 1006, p. 4).

396. Chen discloses that times are represented as Java timestamps internally and are part of the overall time system discussed with respect to limitation 1[d].

Time

Parsing of the time field of a log and insertion of the corresponding time value into the database takes place in three phases... ***Phase 2 involves converting the date time string into a Java type, in particular into a Java SQL TimeStamp***... The stringified representation of this is inserted into the Time field of the Canonical table while the binary representation is inserted into one of the two event-time related tables as starttime and stoptime. That table is keyed by Eventid and lists the event times of all events in the Canonical table. All querying is based on the starttime value.

(Ex. 1006, p. 4, emphasis added).

397. A POSITA would have understood issuing an SQL query with a timestamp to be issuing a timestamp.

398. Thus, Chen discloses limitation 1[e].

6. Limitation 1[f]

399. Limitation 1[f] of the ‘244 patent reads:

identifying, based upon the time of interest, indicia of pestware, wherein the identifying is initiated by the issuing the timestamp; and.

400. Chen discloses limitation 1[f] of the ‘244 patent.

401. Chen discloses providing forensic data in order to identify pestware (in this case, software that elevates Windows privileges). The system identifies the events that are associated with the pestware but all of them are based on the issuing of timestamps discussed with respect to limitation 1[e]. That is, all of these events that are indicia of pestware are identified starting with if they are within the time windows demarcated by the timestamps. Chen discusses this process of identifying indicia of pestware within the time window in detail:

The investigator would presumably be interested in knowing times when the administrator account on the workstation was logged in to. The ECF tool Dynamic Query interface was used to generate the following query which will return a list of records relating to logins to the ‘administrator’ account on the workstation in question...

This query returned a single record from our test logs which we do not show here due to space limitations. However, the record returned occurred at the time ‘2003-08-08 04:56:5’ (UCT). The investigator would also be interested in other accounts that had been used to login to the workstation at around the same time. ***Therefore, in a similar fashion to the previous scenario we perform a query to determine all the users who logged onto the workstation ‘dsto’ in the time window commencing ten minutes prior to the identified records, and finishing ten minutes after it...***

This query returned the three entries shown below in Figure 3... ***The investigator would presumably then be interested in discovering more about the activities of that user in the same time-frame...***

In fact, ***we perform a query which shows all of the activity that occurred on the workstation in the chosen time window...***

This query returns 122 records which we do not show here. Through a manual inspection of these logs we notice that the workstation restarted at ‘2003-08-08 04:56:28’... ***We now decide to investigate other logs available to us in order to try to gain more information above the activities of ‘bob’ in the chosen time period. A similar query is performed, this time without the LOGNODE = ‘dsto’ restriction. Figure 4 below shows the results of the query with all rows hidden except those associated with the files downloaded from the web server by the ‘dsto’ workstation. We see that the workstations IP address (131.181.6.167) appears in the Apache logs and***

observe that a number of suspicious looking files have been downloaded by 'bob'.

The files 'rawrite2.exe' and 'bd030426.bin' actually relate to the exploit in this case. The 'rawrite2.exe' utility is a tool for copying bootable floppy disk images to a floppy disk. The image file, in this case 'bd030426.bin', contains a utility for wiping the Windows administrator password (on vulnerable Windows 2000 hosts). *These logs give us a very strong indication of what actually happened in this case.*

(Ex. 1006, pp. 8-9, emphasis added).

402. Chen also discloses that an automated process could perform this kind of analysis of the data:

Note that it is our intention that the ECF tool will eventually be able to make many of the inferences that we currently perform manually, by itself. It is envisaged that data mining approaches, combined with the ability to relate the additional log-specific information between the different log types, will allow the tool to extract only the most relevant logs, leaving extraneous ones out. This problem is analogous to the current problem with intrusion detection systems which can generate very large amounts of alerts, many of which are not relevant.
(Ex. 1006, p. 9, emphasis added).

403. Thus, Chen discloses limitation 1[f].

7. Limitation 1[g]

404. Limitation 1[g] of the '244 patent reads:

accessing, using a hardware processor of the computer, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of an externally networked source that is suspected of originating pestware.

405. Chen in view of Piccard discloses limitation 1[g] of the '244 patent.

406. As discussed with respect to limitation 1[f], Chen discloses identifying indicia of pestware. In the examples I provided, specific files names are identified as potentially being pestware. Moreover, Chen discussed using web browser logs, which would be at least a portion of a recorded history of externally networked sources that the computer received files from.

The browser logs located on workstation W could be used to gain additional information about the different sites that the user visited, however we have not included those logs in this particular scenario as the browser cache did not contain a copy of the exploit file (due to restrictions on the type/size of files stored in the cache by browsers, as described above). (Ex. 1006, p. 8).

407. Chen explicitly disclosed using browser logs (e.g., Apache files) in identifying potential malware: “We see that the workstations IP address (131.181.6.167) appears *in the Apache* logs and observe that a number of suspicious looking files have been downloaded by ‘bob’.” (Ex. 1006, p. 8, emphasis added).

408. Piccard also teaches using a download history log, which is at least a portion of a recorded history of externally networked sources, to identify the external source (e.g., URL) of a file suspected of being malware:

Embodiments of the invention include systems of managing malware. In one embodiment, a system includes a malware detection module configured to analyze a file of a protected computer to determine that the file is associated with malware. The system also includes a Web site identification module configured to search a download history log of the protected computer to identify a Web site from which the file was downloaded. (Ex. 1007, [0005]).

409. In more detail, Piccard discloses that the “Web site identification module” is invoked after a determination of malware by the “malware detection module”. A POSITA would have found it obvious to have the “malware detection module” either be or be combined with Chen’s ECF as described in limitation 1[f]. Chen disclosed identifying potential malware including the specific names of files. A POSITA would have understood a component that

identifies malware to be a “malware detection module.” Thus, it would have been obvious to have Chen’s ECF trigger Piccard’s “Web site identification module.” Piccard indicates that the “Web site identification module” then accesses the history log to determine the external source:

In particular, once a file is downloaded using the Web browser 114, the malware detection module 118 analyzes the file to determine whether the file includes potential malware. ***If the file is determined to include potential malware, the Web site identification module 120 determines a Web address from which the file was received.*** In particular, the Web site identification module 120 ***accesses the history log 116 to identify the Web address of the file.*** In such manner, the Web site identification module 120 can identify a Web site from which the file was downloaded and, thus, can identify the Web site as a potential malware distribution site. (Ex. 1007, [0015]).

410. Piccard’s “a Web address from which the file was received” teaches the claimed “so as to identify, based at least in part upon the identified indicia of pestware, a reference to an identity of an externally networked source that is suspected of originating pestware”.

411. As I discussed in Section IX.A., a POSITA would have been motivated to combine Chen with Piccard. Chen is directed to the identification of pestware and recognizes the efficiency and accuracy gained through correlating multiple sources of activity information in that identification. Chen further teaches the need to correlate information on a network level using, for example Apache logs, a concept taught by Piccard’s identification and reporting modules. Piccard teaches identifying the origin of pestware (e.g., a URL) and sharing that information with a research entity in order to provide better detection capabilities to nodes on the network. Furthermore, Piccard teaches a “malware detection module” with various exemplary techniques for detecting malware. Chen’s techniques are directed to the same purpose as Piccard’s “malware detection module”, and a POSITA would have expected a high likelihood of success in combining those teachings.

412. Thus, Chen in view of Piccard discloses limitation 1[g].

8. Limitation 1[h]

413. Limitation 1[h] of the '244 patent reads:

wherein the recorded history of externally networked sources is stored on the file storage device.

414. Chen in view of Piccard discloses limitation 1[h] of the '244 patent.

415. Chen discloses examining data in Apache log files. The Apache server log files are not originally stored locally on the ECF's device storage. However, the ECF copies the log data into its database, which is stored locally on the ECF's system.

[The Log Parse module] provides the interface to all log parser software which in addition to parsing the logs (in some cases preparsed), inserts the event canonical form representation ***into the database***. This involves insertion of the base or canonical event information into the Canonical table, and insertion of related information into the Logrecord and time-related tables.

(Ex. 1006, p. 4, emphasis added).

416. See also, e.g., Chen's Figure 1, reproduced hereinbelow:

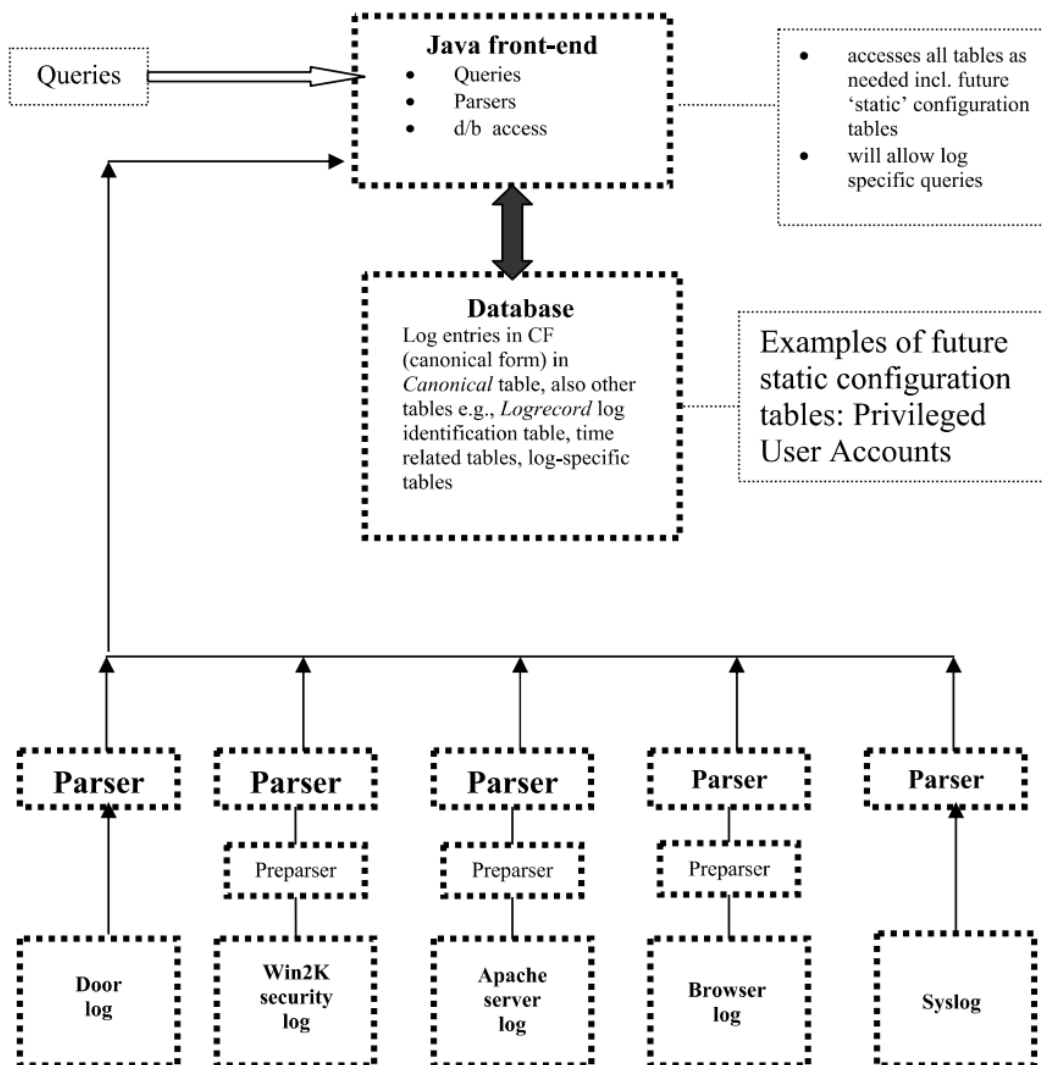


Figure 1: The ECF Architecture

417. Piccard discloses using web browser download logs. A POSITA would have known that typically browsers store their download logs on the local file storage device. Piccard also discloses this explicitly:

As illustrated in FIG.1, **the memory 112 also stores a history log 116**, which is maintained by the Web browser 114 to provide a record of browsing events. **In particular, when a file is accessed and downloaded, the Web browser 114 records a Web address of the file in the history log 116.** (Ex. 1007, [0013], emphasis added).

In particular, the Web site identification module 120 accesses the history log 116 to identify the Web address of the file. In such manner, the Web site identification module 120 can identify a Web site from which the file was downloaded and, thus, can identify the Web site as a potential malware distribution site.
(Ex. 1007, [0015], emphasis added).

418. I have already explained why a POSITA would have been motivated to combine Chen in view of Piccard in Section X.A., and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

419. Thus, Chen in view of Piccard discloses limitation 1[h].

420. Thus, I conclude that Chen in view of Piccard renders Claim 1 obvious to a POSITA.

C. Claim 2 is rendered obvious by Chen in view of Piccard

421. Claim 2 of the ‘244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Chen in view of Piccard. Each of the limitations of Claim 1 of the ‘244 patent is rendered obvious by Chen in view of Piccard as described below.

1. Limitation 2[a]

422. Limitation 2[a] of the ‘244 patent reads:

The method of claim 1, wherein the identifying includes: accessing at least one log of historical events on the computer

423. Chen discloses the additional requirements of limitation 2[a].

424. Chen discloses accessing logs that store historical events and consolidating them into a database:

The research described in this paper focuses on the nature of the event information provided in commonly available computer and other logs (such as system event logs, audit logs, door logs etc) and the extent to which it is possible to correlate such event information despite its heterogeneous nature and origins. The strategic purpose of the research has been to develop

a means by which a consolidated repository of such event information can be constituted and then queried in order to provide an investigator with post hoc event correlation for forensics purposes (ECF).
(Ex. 1006, p. 1).

425. The data is accessed when it is pulled from log files and merged into the database.

Chen discloses taking each log entry and normalizing the format to a uniform representation.

The decision was made to opt for the former viz., for a canonical representation of what is essentially the minimal or base set of fields which appear in all event records viz., Time, Subject, Action, and Object. These fields map intuitively to almost all event records no matter how varied the events or systems on which they occur, and so a single database table (the Canonical table) which represents that information is consequently the central information store used for event correlation in ECF.
(Ex. 1006, p. 2).

426. Moreover, Chen describes querying the database representing the aggregated log file during event investigation:

This scenario relies on only three log types: the door log; the Windows 2000 security log; and the Apache web server log... The following SQL query, which was generated using the Dynamic Query interface, extracts the desired records: ... This query returned the three entries shown below in Figure 3.
(Ex. 1006, p. 8).

427. Thus, Chen in view of Piccard discloses limitation 2[a].

2. Limitation 2[b]

428. Limitation 2[b] of the '244 patent reads:

locating at least one event in the at least one log that corresponds to the time of interest; and

429. Chen discloses the additional requirements of Limitation 2[b].

430. Chen discloses locating events that correspond to the time of interest by identifying events that are within a time interval of the time of interest.

431. Chen discloses that the SQL queries can specify a time of interest in the user interface (e.g., time windows around a specific event as discussed below):

The investigator would presumably be interested in knowing times when the administrator account on the workstation was logged in to. ***The ECF tool Dynamic Query interface was used to generate the following query*** which will return a list of records relating to logins to the ‘administrator’ account on the workstation in question (‘dsto’)...

This query returned a single record from our test logs which we do not show here due to space limitations. ***However, the record returned occurred at the time ‘2003-08-08 04:56:5’ (UCT).*** The investigator would also be interested in other accounts that had been used to login to the workstation at around the same time. Therefore, in a similar fashion to the previous scenario ***we perform a query to determine all the users who logged onto the workstation ‘dsto’ in the time window commencing ten minutes prior to the identified records,*** and finishing ten minutes after it. The following SQL query, which was generated using the Dynamic Query interface, extracts the desired records:

```
SELECT      *      FROM      CANONICAL,      LOGRECORD      WHERE
CANONICAL.LOGID      =      LOGRECORD.LOGID      AND
CANONICAL.TIME      >=      '2003-08-08      04:46:51'      AND
CANONICAL.TIME      <=      '2003-08-08      05:06:51'      AND
CANONICAL.ACTION = 'Logon' AND LOGRECORD.LOGNODE =
'dsto';
```

(Ex. 1006, p. 8, emphasis added).

432. Chen’s indication of a time of interest is a time interval as discussed above (e.g., the time windows commencing ten minutes prior to the identified records).

433. Chen discloses a “Dynamic Query interface” as an interface to the system. The Dynamic Query interface permits users to select the data from the database graphically and using constraints.

Dynamic queries are satisfied largely by access to the main database table, the Canonical table with processing of time values relying upon the field starttime in one of the two event-time related tables...

In presenting a query, the operator has the option of selecting a subset of the Canonical fields to be displayed in the results by setting the appropriate checkboxes.

The query is built by selecting various constraints and relating them with AND and/or OR relationships. For example, if both Time and Subject are constraints, using the AND relationship means both Time and Subject constraints need to be satisfied, using the OR relationship means that the result set should include events that satisfy either constraint.

When a constraint is selected, the application populates the related drop down menu with all corresponding values currently in the database and this may then be used to simply choose the desired value as the constraint. The constraints can also be set to match a sub-string. For example, if SubjectType is set to ip_address and Subject is set to “contains 131.181” then all IP addresses in the 131.181.0.0 subnet will be matched and only records satisfying that constraint will be displayed (subject to other constraints also).

(Ex. 1006, p. 4).

434. A POSITA would have understood that the timing constraints identified by Chen above (e.g., “within 10 minutes”) are constraints that could be expressed by the constraints of the Dynamic Query interface.

435. Chen also discloses identifying events in the log that correspond to the time interval of the time of interest. Chen discloses that the system identifies the events based on the interval specified in the Dynamic Query interface as discussed above. Chen discusses this process of identifying the events within the time window in detail:

The investigator would presumably be interested in knowing times when the administrator account on the workstation was logged in to. The ECF tool Dynamic Query interface was used to generate the following query which will return a list of records relating to logins to the ‘administrator’ account on the workstation in question...

This query returned a single record from our test logs which we do not show here due to space limitations. However, the record returned occurred at the time ‘2003-08-08 04:56:5’ (UCT). The investigator would also be interested in other accounts that had been used to login to the workstation at around the same time. ***Therefore, in a similar fashion to the previous scenario we perform a query to determine all the users who logged onto the workstation ‘dsto’ in the time window commencing ten minutes prior to the identified records, and finishing ten minutes after it...***

This query returned the three entries shown below in Figure 3... *The investigator would presumably then be interested in discovering more about the activities of that user in the same time-frame...*

In fact, *we perform a query which shows all of the activity that occurred on the workstation in the chosen time window...*

This query returns 122 records which we do not show here. Through a manual inspection of these logs we notice that the workstation restarted at '2003-08-08 04:56:28'... *We now decide to investigate other logs available to us in order to try to gain more information above the activities of 'bob' in the chosen time period. A similar query is performed, this time without the LOGNODE = 'dsto' restriction. Figure 4 below shows the results of the query with all rows hidden except those associated with the files downloaded from the web server by the 'dsto' workstation. We see that the workstations IP address (131.181.6.167) appears in the Apache logs and observe that a number of suspicious looking files have been downloaded by 'bob'.*

(Ex. 1006, pp. 8-9, emphasis added).

436. Chen also discloses that an automated process could perform this kind of analysis of the data:

Note that it is our intention that the ECF tool will eventually be able to make many of the inferences that we currently perform manually, by itself. It is envisaged that data mining approaches, combined with the ability to relate the additional log-specific information between the different log types, will allow the tool to extract only the most relevant logs, leaving extraneous ones out. This problem is analogous to the current problem with intrusion detection systems which can generate very large amounts of alerts, many of which are not relevant.

(Ex. 1006, p. 9, emphasis added).

437. Thus, Chen in view of Piccard discloses limitation 2[b].

3. Limitation 2[c]

438. Limitation 2[c] of the '244 patent reads:

determining that the at least one event is associated with the indicia of pestware.

439. Chen discloses the additional requirements of limitation 2[c].

440. Chen discloses that the events identified within the time window can be determined to be indicia of malicious software (pestware).

This query returns 122 records which we do not show here. Through a manual inspection of these logs we notice that the workstation restarted at '2003-08-08 04:56:28'... ***We now decide to investigate other logs available to us in order to try to gain more information above the activities of 'bob' in the chosen time period. A similar query is performed, this time without the LOGNODE = 'dsto' restriction. Figure 4 below shows the results of the query with all rows hidden except those associated with the files downloaded from the web server by the 'dsto' workstation. We see that the workstations IP address (131.181.6.167) appears in the Apache logs and observe that a number of suspicious looking files have been downloaded by 'bob'.***

The files 'rawrite2.exe' and 'bd030426.bin' actually relate to the exploit in this case. The 'rawrite2.exe' utility is a tool for copying bootable floppy disk images to a floppy disk. The image file, in this case 'bd030426.bin', contains a utility for wiping the Windows administrator password (on vulnerable Windows 2000 hosts). These logs give us a very strong indication of what actually happened in this case.

(Ex. 1006, pp. 8-9, emphasis added).

441. Thus, Chen in view of Piccard discloses limitation 2[c].

442. Thus, I conclude that Chen in view of Piccard renders Claim 2 obvious to a POSITA.

D. Claim 3 is rendered obvious by Chen in view of Piccard

443. Claim 3 of the '244 patent, which depends from Claim 2, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 3 reads,

The method of claim 2, wherein the determining that the at least one event is associated with the indicia of pestware includes following references in the at least one log to a suspected pestware object..

444. Chen discloses the additional requirements of Claim 3.

445. Chen discloses identifying a suspected malware file, which is a pestware object, based on data within the log files.

The files ‘rawrite2.exe’ and ‘bd030426.bin’ actually relate to the exploit in this case. The ‘rawrite2.exe’ utility is a tool for copying bootable floppy disk images to a floppy disk. The image file, in this case ‘bd030426.bin’, contains a utility for wiping the Windows administrator password (on vulnerable Windows 2000 hosts). ***These logs give us a very strong indication of what actually happened in this case.***

(Ex. 1006, p. 9, emphasis added).

446. Thus, I conclude that Chen in view of Piccard renders Claim 3 obvious to a POSITA.

E. Claim 4 is rendered obvious by Chen in view of Piccard

447. Claim 4 of the ‘244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 4 reads,

The method of claim 3, wherein following the references includes following at least one reference in the activity log from the event to a suspected pestware process.

448. Chen in view of Piccard discloses the additional requirements of Claim 4.

449. As I explained with respect to Claim 3, Chen discloses identifying file objects associated with the suspected pestware. A POSITA would have known that a “process” is an executable file that is running.

450. Piccard teaches analyzing the file identified by a malware detection device including whether or not the file acts suspiciously. Piccard explains:

As a further example, the set of malware definitions can include suspicious activities related to third-party cookies or related to entries or modifications of registry files of an operating system.
(Ex. 1007, [0020]).

451. A POSITA would have understood that only a file that is executing (i.e., is a process) is capable of suspicious activities such as modification to the registry file of an operating system. In combination with Chen, the log data that identifies the file is also combined with the activities of the associated process.

452. As I discussed in Section X.A., a POSITA would have been motivated to combine Chen and Piccard. I incorporate my analysis herein by reference.

453. Thus, I conclude that Chen in view of Piccard renders Claim 4 obvious to a POSITA.

F. Claim 5 is rendered obvious by Chen in view of Piccard

454. Claim 5 of the '244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 5 reads,

The method of claim 2, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

455. Chen discloses the additional requirements of Claim 5.

456. I have already explained with respect to limitation 1[c] that the Windows 2000 Security Log, and other logs used by Chen, are activity logs. As also explained with respect to limitation 1[c], these logs are also event logs.

457. Chen also discloses browser logs and browser cache logs. Chen explains that the tested system included “front-end parsers for... Browser Cache Logs” and “Pre-parsing... for... Browser Logs.” (Ex. 1006, p. 5).

458. Thus, I conclude that Chen in view of Piccard renders Claim 5 obvious to a POSITA.

G. Claim 6 is rendered obvious by Chen in view of Piccard

459. Claim 6 of the '244 patent, which depends from Claim 1, is rendered obvious by Chen in view of Piccard. Claim 6 reads:

The method of claim 1, wherein the user input is in response to an alert provided to the user regarding a detected suspicious activity on the computer.

460. Chen in view of Piccard discloses the additional requirements of Claim 6.

461. Chen discloses that investigators are interested in particular events and scenarios, which a POSITA would understand to be initiated or identified by an alert:

Section 4 outlines a number of 'scenarios' - event sequences [such as malware] that may typically be of interest to an investigator - and how the ECF software identifies these scenarios and provides investigators with the means to query and investigate the nature of the events and the circumstances surrounding them.
(Ex. 1006, p. 2).

462. Chen also discloses the use of forensic operations in response to a suspicious event. For example, an unexpected password change may be suspicious and this can trigger an investigation. As Chen explains:

In this case the investigator may have very little idea about the exact nature of the scenario above, but will most likely become suspicious upon noticing that the Administrator password has changed. This discovery may not occur until some time after the exploit was actually carried out.
(Ex. 1006, p. 8).

463. To the extent Chen does not explicitly disclose that the operator did not begin an investigation because of an alert, or had not "noticed" the password change because of an alert, it would have been obvious to a POSITA to do so. Alerts are extremely common for suspicious

activities. For example, Chen explains that “alert correlation” is related work. (*See, e.g.*, Chen, pp. 1, 10).

464. Moreover, Piccard also discloses an alert mechanism when malware is suspected:

In the illustrated embodiment, once the Web site identification module identifies the Web site, a reporting module (e.g., the reporting module 122) reports information regarding the Web site to a remotely-located computer that is connected to the protected computer. This information can identify the Web site as a potential malware distribution site, such as in terms of the domain name of the Web site. Alternatively, or in conjunction, this information can identify the file as including potential malware, Such as in terms of the name of the file or the URL of the file. It is also contemplated that this information can include a representation of the file or can identify suspicious activities related to the file.

In the illustrated embodiment, the reporting module also alerts a user of the protected computer about the Web site.
(Ex. 1007, [0023]-[0024]).

465. To the extent Chen does not disclose this limitation alone, it would have been obvious to a POSITA in view of Piccard’s disclosures. I have already provided a general motivation to combine Chen and Piccard in Section X.A. and I incorporate my analysis into this section by reference. Furthermore, a POSITA would have found it obvious to notify forensic investigators, as disclosed by Chen, when abnormal information was identified. Chen already teaches responding to potential problems (e.g., password changes) so to the extent it is not automated or not through an alert system, it would have been obvious to do so.

466. Thus, I conclude that Chen in view of Piccard renders Claim 6 obvious to a POSITA.

H. Claim 7 is rendered obvious by Chen in view of Piccard

467. Claim 7 of the ‘244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 7 reads,

The method of claim 1, including: reporting the identity of the externally networked source to an externally networked pestware research entity so as to enable the pestware research entity to research whether the externally networked source is a source of pestware.

468. Chen in view of Piccard discloses this Claim.

469. As discussed with respect to limitation 1[h], Piccard discloses identifying the identity of the externally networked source. Piccard also discloses reporting this information to a remotely-located computer. Piccard explains:

The third operation illustrated in FIG. 2 is to report that the Web site is a potential malware distribution site (block 204). In the illustrated embodiment, once the Web site identification module identifies the Web site, a reporting module (e.g., the reporting module 122) reports information regarding the Web site ***to a remotely-located computer*** that is connected to the protected computer. This information can identify the Web site as a potential malware distribution site, such as in terms of the domain name of the Web site... This information as well as any additional relevant ***information can be analyzed at the remotely-located computer to determine whether the file does, in fact, include malware*** and whether the Web site is, in fact, a malware distribution site.
(Ex. 1007, [0023], emphasis added).

470. A POSITA would have understood that a “remotely-located computer” that “analyze[s]... to determine whether ... the Web site is, in fact, a malware distribution site ” is an externally networked pestware research entity. This “remotely-located computer” serves the purpose of analyzing the information to determine whether the website is indeed a source of malware and subsequently reporting its findings. The “reporting module” is responsible for transmitting the necessary information to do such analysis.

471. I have already explained why a POSITA would be motivated to combine Chen and Piccard in Section X.A. and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

472. Thus, I conclude that Chen in view of Piccard renders Claim 7 obvious to a POSITA.

I. Claim 8 is rendered obvious by Chen in view of Piccard

473. Claim 8 of the '244 patent, which depends from Claim 1, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 8 reads,

The method of claim 1, wherein the externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

474. Chen in view of Piccard discloses Claim 8.

475. As explained with respect to limitation 1[g], Piccard discloses identifying the external network source of the potential pestware. Piccard discloses identifying the source by both IP address and URL.

As illustrated in FIG.1, the memory 112 also stores a history log 116, which is maintained by the Web browser 114 to provide a record of browsing events. In particular, when a file is accessed and downloaded, the Web browser 114 records a Web address of the file in the history log 116. A Web address typically specifies a location of a file within a Web site. ***For example, a Web address can be a Uniform Resource Identifier ("URI) of a file, such as a Uniform Resource Locator ("URL) of the file. It is also contemplated that a Web address can be defined in various other ways, such as using an Internet Protocol ("IP) address or any other identifier of a source of a file.***
(Ex. 1007, [0013], emphasis added).

476. I have already explained why a POSITA would be motivated to combine Chen and Piccard in Section X.A. and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

477. Thus, I conclude that Chen in view of Piccard renders Claim 8 obvious to a POSITA.

J. Claim 9 is rendered obvious by Chen in view of Piccard

478. Each of the limitations of Claim 9 of the ‘244 patent is rendered obvious by Chen in view of Piccard as described below.

1. Limitation 9[a]

479. Limitation 9[a] of the ‘244 patent reads:

A system for identifying a source of suspected pestware on a computer, the system comprising:

480. I understand that the preamble of a patent claim is not always considered limiting. However, to the extent that the preamble of Claim 9 of the ‘244 patent is considered limiting, Chen and Piccard both disclose the preamble of Claim 9.

481. Chen discloses a system of enabling and improving forensic analysis of computer systems and software:

The strategic purpose of the research has been to develop a means by which a consolidated repository of such event information can be constituted and then queried in order to provide an investigator with post hoc event correlation for forensics purposes (ECF).
(Ex. 1006, p. 1).

482. Moreover, Chen explains that the purpose of the ECF is to identify malware, such as an exploit that attacks the Windows operating system: “... scenarios described in this section have been identified to demonstrate the capabilities of the ECF software... [such as the] [d]ownload and execution of an exploit allowing elevated privileges on a Windows 2000 workstation.” (Ex. 1006, p. 6). As discussed in Section V.E., a POSITA would have understood that “elevat[ing] privileges” is a means of installing malware software deep inside the Windows

operating system in order to spy on the system without detection. Thus, Chen discloses identifying suspected pestware activity.

483. Moreover, Chen discloses identifying an origin of the suspected activity. For example, Chen discloses tracing the source of pestware back to the download URL:

The browser logs located on workstation W could be used to gain additional information about the different sites that the user visited, however we have not included those logs in this particular scenario as the browser cache did not contain a copy of the exploit file (due to restrictions on the type/size of files stored in the cache by browsers, as described above).
(Ex. 1006, p. 8).

484. See also, e.g., Figure 1 reproduced hereinbelow:

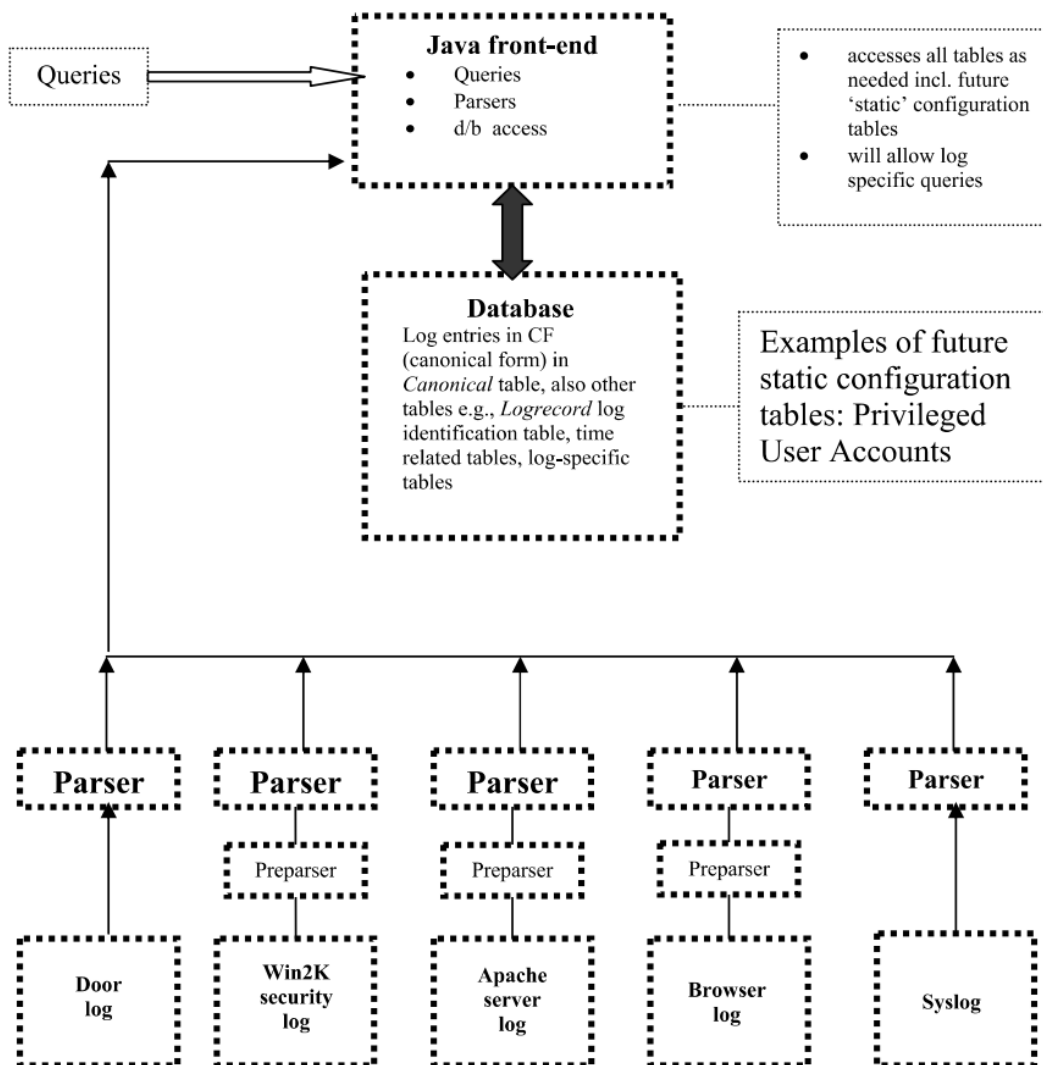


Figure 1: The ECF Architecture

485. Piccard also teaches a system for identifying the source of malware by examining web traffic history: “Embodiments of the invention include systems of managing malware. In one embodiment, a system includes a malware detection module configured to analyze a file of a protected computer to determine that the file is associated with malware. The system also includes a Web site identification module configured to search a download history log of the protected computer to identify a Web site from which the file was downloaded.” (Ex. 1007, [0005]).

486. As I discussed in Section X.A., a POSITA would have been motivated to combine Chen and Piccard. Chen is directed to the identification of processes exhibiting abnormal behavior and anticipates the need for appropriate responses to such processes. Piccard teaches identifying the origin of pestware that has been detected by techniques like those taught in Chen. Chen's techniques are directed to the same purpose as Piccard's "malware detection module", and a POSITA would have expected a high likelihood of success in combining those teachings.

487. Thus, both Chen and Piccard disclose limitation 9[a].

2. Limitation 9[b]

488. Limitation 9[b] of the '244 patent reads:

a hardware processor:

489. Both Chen and Piccard disclose limitation 9[b].

490. Chen discloses a system that makes use of Java and a Postgres database. A POSITA would have known that software (like Java and SQL) requires a processor. Chen explains that the implementation of the system in these components: "The ECF design elaborated in the previous section has been implemented in Java using a Postgres database accessed via JDBC." (Ex. 1006, p. 4).

491. Piccard also discloses a processor (CPU). "In the illustrated embodiment, the protected computer 102 is a client computer and includes conventional client computer components, including a Central Processing Unit ("CPU") 108 that is connected to a network connection device 110 and a memory 112." (Ex. 1007, [0012]).

492. Thus, both Chen and Piccard disclose limitation 9[b].

3. Limitation 9[c]

493. Limitation 9[c] of the '244 patent reads:

a file storage device coupled to the hardware processor:

494. Both Chen and Piccard disclose limitation 9[c].

495. Chen discloses a system that makes use of Java and a Postgres database. A POSITA would have known that software (like Java and SQL) require a file storage device, such as a hard drive. Java requires a file storage system in order to store the Java files for execution. Postgres also requires a file storage device in order to save the database file(s). Chen explains that the implementation of the system in these components: “The ECF design elaborated in the previous section has been implemented in Java using a Postgres database accessed via JDBC.” (Ex. 1006, p. 4). It would have been obvious to a POSITA that in order to execute the Java files or to save data to the Postgres database, the file storage device would be coupled to the hardware processor.

496. Piccard also discloses a file storage device that is coupled to the hardware processor:

An embodiment of the invention relates to a computer program product with a computer-readable medium including computer code or executable instructions thereon for performing a set of computer-implemented operations. The medium and computer code can be those specially designed and constructed for the purposes of the invention, or they can be of the kind well known and available to those having ordinary skill in the computer Software arts. ***Examples of computer-readable media include: magnetic media such as hard disks, floppy disks, and magnetic tape; optical media such as Compact Disc-Read Only Memories (“CD-ROMs”) and holographic devices; magneto-optical media Such as floptical disks;*** and hardware devices that are specially configured to store and execute computer code, such as Application-Specific Integrated Circuits (ASICs), Programmable Logic Devices (“PLDs”), Read Only Memory (“ROM”) devices, and Random Access Memory (“RAM”) devices. (Ex. 1007, [0026], emphasis added).

497. Thus, both Chen and Piccard disclose limitation 9[c].

4. Limitation 9[d]

498. Limitation 9[d] of the ‘244 patent reads:

a timestamp module configured to be executed by the hardware processor, to generate a time of interest including a time interval in response to detected activity on the computer that is indicative of pestware, to issue a timestamp after generating the time of interest, and to initiate, in response to the issuance of the timestamp, identification of indicia of pestware:

499. Chen discloses limitation 9[d].

500. The language of limitation 9[d] is almost identical to the language of limitations 1[d], 1[e], and 1[f]. The only significant difference is that limitation 9[d] requires a timestamp “module” to perform the operations. I incorporate my analysis of limitations 1[d], 1[e], and 1[f] into this section by reference.

501. Chen also discloses a timestamp module. Chen discloses that the ECF Dynamic Query module issues timestamps related to suspicious events in order for the system to find all events within the time period.

This query returned a single record from our test logs which we do not show here due to space limitations. ***However, the record returned occurred at the time ‘2003-08-08 04:56:5’ (UCT).*** The investigator would also be interested in other accounts that had been used to login to the workstation at around the same time. Therefore, in a similar fashion to the previous scenario ***we perform a query to determine all the users who logged onto the workstation ‘dsto’ in the time window commencing ten minutes prior to the identified records,*** and finishing ten minutes after it. The following SQL query, which was generated using the Dynamic Query interface, extracts the desired records:

(Ex. 1006, p. 8, emphasis added).

Dynamic queries are satisfied largely by access to the main database table, the Canonical table with processing of time values relying upon the field starttime in one of the two event-time related tables...

When a constraint is selected, the application populates the related drop down menu with all corresponding values currently in the database and this

may then be used to ***simply choose the desired value as the constraint***. The constraints can also be set to match a sub-string. For example, if SubjectType is set to ip_address and Subject is set to “contains 131.181” then all IP addresses in the 131.181.0.0 subnet will be matched and only records satisfying that constraint will be displayed (subject to other constraints also).

(Ex. 1006, p. 4).

502. Chen discloses that times are represented as Java timestamps internally: “Phase 2 involves converting the date time string into a Java type, in particular into a Java SQL TimeStamp.” (Ex. 1006, p. 4).

503. I have already explained why a POSITA would be motivated to combine Chen and Piccard in Section X.A. and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

504. Thus, Chen discloses limitation 9[d].

5. Limitation 9[e]

505. Limitation 9[e] of the ‘244 patent reads:

a research portion configured to be executed by the hardware processor and to access at least one log on a file storage device of the computer to relate the time of interest to at least one externally networked source of potential pestware activity on the computer;;

506. Chen in view of Piccard discloses limitation 9[e].

507. The language of limitation 9[e] is similar to the language of Claim 2. The only significant difference is “a research portion” that performs the operations.

508. A POSITA would have understood the ECF components that access a log file combined with the Web site identification module that identifies the external networked source to be a “research portion” of the system. These components are executed by the hardware processor

(as they are part of the software), access the file storage device of the computer, and related the time of interest to an externally networked source of potential pestware activity.

509. I have already explained why a POSITA would be motivated to combine Chen and Piccard in Section X.A. and with respect to limitation 1[g]. I incorporate both of these analyses into this section by reference.

510. Thus, Chen in view of Piccard discloses limitation 9[e].

6. Limitation 9[f]

511. Limitation 9[f] of the ‘244 patent reads:

a kernel-mode driver configured to monitor activity on the computer and to generate an activity log that is included within the at least one log; and:

512. Chen discloses limitation 9[f].

513. The language of limitation 9[f] is similar to the language of limitation 1[b]. There are no substantive changes. I incorporate my analysis of limitation 1[b] into this section by reference.

514. Thus, Chen discloses limitation 9[f].

7. Limitation 9[g]

515. Limitation 9[g] of the ‘244 patent reads:

a reporting portion configured to be executed by the hardware processor and to generate a report that identifies the at least one externally networked source of potential pestware activity:

516. Chen in view of Piccard discloses limitation 9[g].

517. The language of limitation 9[g] is similar to limitation 1[g] with respect to identifying the at least one externally networked source of potential pestware activity. The only

significant change is that the operations must be performed by “a reporting portion” and that the reporting portion must “generate a report.” I incorporate my analysis of limitation 1[g] by reference.

518. Piccard discloses generating a report.

The third operation illustrated in FIG. 2 is to report that the Web site is a potential malware distribution site (block 204). In the illustrated embodiment, once the Web site identification module identifies the Web site, ***a reporting module (e.g., the reporting module 122) reports information regarding the Web site to a remotely-located computer that is connected to the protected computer.*** This information can identify the Web site as a potential malware distribution site, such as in terms of the domain name of the Web site. Alternatively, or in conjunction, this information can identify the file as including potential malware, such as in terms of the name of the file or the URL of the file. It is also contemplated that this information can include a representation of the file or can identify suspicious activities related to the file. ***This information as well as any additional relevant information can be analyzed at the remotely-located computer to determine whether the file does, in fact, include malware and whether the Web site is, in fact, a malware distribution site.*** If the Web site is determined to be a malware distribution site, a new or updated set of malware definitions can be generated based on content within the Web site, and the new or updated set of malware definitions can be provided to the protected computer. In addition, content within the Web site can be monitored on a periodic or other basis for new or updated malware. Also, a new or updated list of malware distribution sites can be generated so as to identify the Web site, and the new or updated list of malware distribution sites can be provided to the protected computer. (Ex. 1007, [0023], emphasis added).

519. A POSITA would have understood the reporting module to be the “reporting portion” and that reporting information by the reporting portion is “generating a report.”

520. Thus, Chen in view of Piccard discloses limitation 1[g].

521. Thus, I conclude that Chen in view of Piccard renders Claim 9 obvious to a POSITA.

K. Claim 10 is rendered obvious by Chen in view of Piccard

522. Claim 10 of the '244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 10 reads,

The system of claim 9, wherein the at least one log includes information that relates the time of interest to an event, and wherein the at least one log includes information that relates the event to a suspected pestware object, and wherein the at least one log includes information that relates the suspected pestware object to the at least one externally networked source of potential pestware activity.

523. Chen in view of Piccard discloses Claim 10.

524. The language of Claim 10 is similar to language found in Claims 2, 4, and 8. I incorporate my analysis of those three claims into this section by reference.

525. Thus, I conclude that Chen in view of Piccard renders Claim 10 obvious to a POSITA.

L. Claim 11 is rendered obvious by Chen in view of Piccard

526. Claim 11 of the '244 patent, which depends from Claim 10, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 11 reads,

The system of claim 10, including: a heuristics module configured to be executed by the hardware processor and to evaluate whether the event is a suspected pestware-related event before the research portion accesses information in the at least one log that relates the event to the suspected pestware object.

527. Chen discloses the additional requirements of Claim 11.

528. Chen discloses that research is underway to expand the ECF to use specification languages and/or pattern recognition to identify pestware from events.

We are now undertaking further work in four related areas: (i) the fully automated recognition of scenarios of interest: this ultimately needs a scenario description or event pattern specification language and a pattern recognition capability within ECF as a means of matching scenario descriptions (event pattern specifications) against stored events;” (Ex. 1006, p. 9).

529. As discussed in Section V.D., a POSITA would have understood that matching events against pattern specifications is an example of a “heuristic” as a means for estimating or guessing as to the classification of anomalous events. As I discussed in Section V.D., a “heuristics-based detection engine” may look for behaviors commonly exhibited by malware in order to identify suspected malware, even when other detection mechanisms would not have discovered it (e.g., a never-before-seen virus that does not yet have a signature in the database of an antivirus scanner). (See Section V.D.; see also Skoudis, pp. 11-12). Moreover, as disclosed in limitation 9[e], the access of the research portion is to identify the external network source. As explained within that limitation, the identification of the external network source is performed after malware detection.

530. Thus, I conclude that Chen in view of Piccard renders Claim 11 obvious to a POSITA.

M. Claim 12 is rendered obvious by Chen in view of Piccard

531. Claim 12 of the ‘244 patent, which depends from claim 11, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 12 reads,

The system of claim 11, wherein the heuristics module determines that the at least one event is suspected to be pestware-related by analyzing other

events in connection with the event, wherein the other events are also related to the time of interest

532. Chen discloses the additional requirements of Claim 12.

533. Chen discloses correlation techniques in order for the ECF to match events together.

(ii) the incorporation into ECF of query mechanisms by which log-specific information in addition to the base information (Time, Subject, Object and Action) is queried and correlated with information from the main Canonical table and the possible use of an XML based approach to do this (this feature will expand the range of queries possible and provide a ‘drill-down’ functionality into the front end);
(Ex. 1006, p. 9).

534. As discussed in Section V.D., a POSITA would have understood that matching events against pattern specifications is an example of a “heuristic” as a means for estimating or guessing as to the classification of anomalous events. As I discussed in Section V.D., a “heuristics-based detection engine” looks for behaviors commonly exhibited by malware in order to identify suspected malware, even when other detection mechanisms would not have discovered it (e.g., a never-before-seen virus that does not yet have a signature in the database of an antivirus scanner). (See Section V.D.; see also Skoudis, pp. 11-12). Moreover, as disclosed in limitation 9[e], the access of the research portion is to identify the external network source. As explained within that limitation, the identification of the external network source is performed after malware detection.

535. Thus, I conclude that Chen in view of Piccard renders Claim 12 obvious to a POSITA.

N. Claim 13 is rendered obvious by Chen in view of Piccard

536. Claim 13 of the ‘244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 13 reads,

The system of claim 9, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history,

browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

537. This claim is virtually identical to Claim 5. I incorporate my analysis of Claim 5 herein by reference and conclude that Chen in view of Piccard renders Claim 13 obvious to a POSITA.

O. Claim 14 is rendered obvious by Chen in view of Piccard

538. Claim 14 of the '244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 14 reads,

The system of claim 9, wherein the timestamp module is further configured to generate the time of interest as a point in time.

539. Chen discloses the additional requirements of Claim 14.

540. I explained with respect to limitation 9[d] that the time of interest includes a time interval. However, Chen also discloses a “point in time” as well. Specifically, Chen discloses identifying a “point in time” of a suspicious event on which the interval of time is based. Chen explains:

The investigator would presumably be interested in knowing times when the administrator account on the workstation was logged in to. ***The ECF tool Dynamic Query interface was used to generate the following query*** which will return a list of records relating to logins to the ‘administrator’ account on the workstation in question (‘dsto’)...

This query returned a single record from our test logs which we do not show here due to space limitations. ***However, the record returned occurred at the time ‘2003-08-08 04:56:5’ (UCT).***

(Ex. 1006, p. 8, emphasis added).

541. Thus, I conclude that Chen in view of Piccard renders Claim 14 obvious to a POSITA.

P. Claim 15 is rendered obvious by Chen in view of Piccard

542. Claim 15 of the '244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 15 reads,

The system of claim 9, wherein the at least one externally networked source of potential pestware activity is identified by an identifier selected from the group consisting of an I.P. address and a URL.

543. This claim is virtually identical to Claim 8. I incorporate my analysis of Claim 8 herein by reference and conclude that Chen in view of Piccard renders Claim 15 obvious to a POSITA.

Q. Claim 16 is rendered obvious by Chen in view of Piccard

544. Claim 16 of the '244 patent, which depends from Claim 9, is rendered obvious to a POSITA by Chen in view of Piccard. Claim 16 reads,

The system of claim 9, wherein the at least one log includes at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

545. This claim is **exactly** identical to Claim 13. I incorporate my analysis of Claim 13 herein by reference and conclude that Chen in view of Piccard renders Claim 16 obvious to a POSITA.

XI. THERE ARE NO OBJECTIVE INDICIA OF NONOBVIOUSNESS

546. I am not aware of any objective indicia of non-obviousness related specifically to the '244 patent. Indeed, I am not aware of any evidence of commercial success, long felt but

unsolved need, failure of others, industry praise, or unexpected results specifically related to the '244 patent.

XII. CONCLUSION

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

Executed on September 7, 2023

By: 
Seth James Nielson