IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

_____

BEFORE THE PATENT TRIAL AND APPEAL BOARD

_____

TREND MICRO, INC.
Petitioner,

v.

TAASERA LICENSING LLC
Patent Owner.

_____

IPR2023-00801

U.S. Patent No. 8,327,441

DECLARATION OF SETH JAMES NIELSON

TABLE OF CONTENTS

I. INTRODUCTION 1

II. QUALIFICATIONS 2

III. MATERIALS CONSIDERED 12

IV. LEGAL STANDARDS 13

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

A. Remote Attestation.....16

VI. OVERVIEW OF THE CHALLENGED PATENT 20

A. THE ’441 PATENT SPECIFICATION AND CLAIMS.....20

B. PROSECUTION HISTORY.....23

VII. LEVEL OF SKILL IN THE ART 23

VIII. Overview of the Prior art 24

A. OVERVIEW OF "INTEGRITY MANAGEMENT FOR TRUSTED COMPUTING" BY MUNETOH, ET. Al ("MUNETOH") 24

B. OVERVIEW OF "TISA: TOWARD TRUSTWORTHY SERVICES IN A SERVICE-ORIENTED ARCHITECTURE" TO RAJAN 28

IX. GROUND 1: Claims 1-7 AND 9 are rendered obvious by MuNETOH IN VIEW OF RAJAN.....34

A. Motivation to Combine Munetoh and Rajan 34

B. Claim 1 is rendered obvious by Munetoh in view of Rajan 40

1. Limitation 1[a].....41

2. Limitation 1[b] 49

3. Limitation 1[c].....53

4. Limitation 1[d] 56

5. Limitation 1[e].....60

6. Limitation 1[f] 64

C. Claim 2 is rendered obvious by Munetoh in view of Rajan 68

1. Limitation 2[a].....68

2. Limitation 2[b] 70

D. Claim 3 is rendered obvious by Munetoh in view of Rajan 72

E. Claim 4 is rendered obvious by Munetoh in view of Rajan 75

F. Claim 5 is rendered obvious by Munetoh in view of Rajan 78

G. Claim 6 is rendered obvious by Munetoh in view of Rajan 80

H. Claim 7 is rendered obvious by Munetoh in view of Rajan 82

I. Claim 9 is rendered obvious by Munetoh in view of Rajan 83

X. Ground 2: Claims 1-7 and 9 are rendered obvious by Rajan in view of Munetoh 84

A. Motivation to Combine Rajan and Munetoh 84

B. Claim 1 is rendered obvious by Rajan in view of Munetoh 85

1. Limitation 1[a].....85

2. Limitation 1[b] 86

3. Limitation 1[c].....86

4. Limitation 1[d] 87

5. Limitation 1[e].....88

6. Limitation 1[f] 89

C. Claim 2 is rendered obvious by Rajan in view of Munetoh 90

1. Limitation 2[a].....90

2. Limitation 2[b] 90

D. Claim 3 is rendered obvious by Rajan in view of Munetoh 91

E. Claim 4 is rendered obvious by Rajan in view of Munetoh 92

F. Claim 5 is rendered obvious by Rajan in view of Munetoh 92

G. Claim 6 is rendered obvious by Rajan in view of Munetoh 93

H. Claim 7 is rendered obvious by Rajan in view of Munetoh 94

I. Claim 9 is rendered obvious by Rajan in view of Munetoh 94

XI. There are no objective indicia of nonobviousness.....95

XII. Conclusion 95

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. 9,092,441 ("the '441 patent") against Taasera Licensing LLC ("Taasera"). All "Ex. 10XX" cites herein are to the Exhibits to the Petitions. 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 '441 patent are obvious in light of the prior art discussed below. The '441 patent is directed to general application of known computer security techniques to remote attestation of security posture.

4. In summary, my conclusion is that the claims of the '441 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 1-7 and 9 are rendered obvious to a POSITA by the combination of "Network Intrusion Detection" by Munetoh et al. ("Munetoh") (Ex. 1005) in view of "Tisa: Toward Trustworthy Services in a Service-Oriented Architecture" by Rajan et al. ("Rajan") (Ex. 1006).

b. Claims 1-7 and 9 are rendered obvious to a POSITA by the combination of "Tisa: Toward Trustworthy Services in a Service-Oriented Architecture" by Rajan et al. ("Rajan") (Ex. 1006) in view of "Network Intrusion Detection" by Munetoh et al. ("Munetoh") (Ex. 1005).

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. Exhibit 1004 is a current copy of my C.V. and 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 '441 patent and its file history, the other documents and references as cited herein, and general technical references.

50. I understand that Counsel for Petitioner have submitted a Petition that references by exhibit number. For simplicity and convenience, I have adopted the same exhibit numbering. The exhibits are:

- Ex. 1001: U.S. Patent No. 8,327,441 to Kumar et al. ("'441 patent")
- Ex. 1002: Prosecution History of '441 to Kumar et al.
- Ex. 1003: Expert Declaration of Seth James Nielson, Ph.D under 37 C.F.R. § 1.68.
- Ex. 1004: Curriculum Vitae of Seth James Nielson, Ph.D.
- Ex. 1005: Seiji Munetoh, Integrity Management Infrastructure for Trusted Computing, IEICE TRANS. INF. & SYST., Vol. E91-D, No. 5, 1242-1251, (May 2008). ("Munetoh")
- Ex. 1006: Hridesh Rajan, Tisa: Toward Trustworthy Services in a Service-Oriented Architecture, IEEE Transactions on Services Computing, 201-213, (October-December 2008). ("Rajan")

- Ex. 1007: Declaration of Ingrid Hsieh-Yee, Ph.D.
- Ex. 1008: Steve Anderson, Web Services Trust Language (WS-Trust), (February 2005).
- Ex. 1009: George Moncrief, ezHPC Security Architecture, IEEE Computer Society, (2006).
- Ex. 1010: Taasera Licensing LLC, v. Trend Micro Incorporated, Plaintiff's Disclosure of Asserted Claims and Infringement Contentions with '441 Claim Chart, 2:21-cv-00441-JRG-RSP, (July 26, 2022).
- Ex. 1011: Ajay Surie, Rapid Trust Establishment for Pervasive Personal Computing, IEEE Computing Society, 24-30, (2007).
- Ex. 1012: Minjin Kwon, PROBE: A Process Behavior-based Host Intrusion Prevention System, Department of Computer Science and Engineering, Korea University, Seoul, (2008).

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 and understand that patent claims are construed from the perspective of one of ordinary skill in the art at the time the claimed invention was made and that, during this proceeding, claims are construed using the same claim construction standard that would be used to construe the claim in a civil action. Under that standard, claim terms are to be given their plain and ordinary meaning, which is the meaning understood by a POSITA in light of the claim language, patent specification, and prosecution history.

53. I understand that a patent may include independent claims and dependent claims. An independent claim stands by itself and only 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 claim includes all of the limitations it recites plus the limitations recited in the claim from which it depends.

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

55. I further understand that the claimed subject matter would have been obvious to one of ordinary skill in the art if, for example, it results from the combination of known elements according to known methods to yield predictable results, the simple substitution of one known element for another to obtain predictable results, the use of a known technique to improve similar devices in the same way, the application of a known technique to a known device ready for improvement to yield predictable results, or choosing from a finite number of identified, predictable solutions with a reasonable expectation of success. I have also been informed that the analysis of obviousness may include recourse to logic, judgment, and common sense available to the person of ordinary skill in the art that does not necessarily require explication in any reference. I understand that so-called "secondary considerations of non-obviousness," must be considered in

an obviousness analysis. I understand that an analysis cannot use hindsight in determining whether a patent claim is obvious. I understand that secondary considerations of non-obviousness include: (a) a long-felt but unresolved need for the invention; (b) commercial success of the invention; (c) copying of the invention; (d) praise and recognition of the invention by others; (e) licensing of the rights to the invention; and (f) unexpected results.

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

57. I understand that a challenged claim can be invalid for obviousness over a combination of prior art references if a reason existed (at the effective filing date of the Patent in question) that would have prompted a person of ordinary skill in the art to combine prior art elements in the way required by the claim in question. I understand that this requirement is sometimes called "motivation to combine." I further understand that the prior art references themselves might provide a motivation to combine, and that sometimes commonsense as would have been known to a POSITA can provide a reason to combine the disclosures of 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 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 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 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 a U.S. patent may be granted an earlier effective filing date if the patent owner properly claimed priority to an earlier patent application.

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 '441 patent that would have been known to a POSITA.

A. Remote Attestation

63. One of the challenges in computer security is how to know if a remote system is trustworthy. For example, if you send sensitive information to a remote system, will it protect it as it is supposed to? Even if the human operators are, themselves, trustworthy, how can someone be sure that their system hasn't been compromised by malware? As explained by Munetoh, one of the prior art references I have used in my analysis:

> Currently, security incidents are frequently reported, such as information leakage of sensitive personal data, swindles based on phishing sites, password leakage by key-loggers, zero-day attacks, to name a few. A primary root cause is malware, involving the installation of malicious

> software which is not recognized by users as being malicious, according to the CS1/FBI Computer Crime and Security Survey 2006 [1]. (Ex. 1005 at p. 1).

64. Rajan, another of the prior art references used in my analysis, describes it another way: clients need to be able to validate "non functional requirements" such as a "service shall not persist [store] the credit card number supplied by the client." (Ex. 1006 at p. 1). Verification of "functional" requirements can be done by "test[ing] the published [external] interface." (*Id.*). Testing the non-functional requirements "may only come from a monitor… that is executing in the same domain as the service implementation…" (*Id.*).

65. One solution to this problem is known as "Remote Attestation." For example, the Trusted Computing Group (TCG) defines a Remote Attestation technique reference model that provides "integrity data for each component running on [a] platform…" (Ex. 1005 at p. 1). The term "integrity," as used within attestation, generally refers to "the state of the components within a computing platform…" (*Id.* at p. 4). In other words, through Remote Attestation, a computer system can determine what kind of software components are running on a remote system. This determination can be compared against some kind of established acceptable set of software. "[T]he ultimate objective of Trusted Computing is to ascertain that the components in a computing platform are in known and trustworthy states, and have not been tampered with." (*Id.* at 4).

66. The general approach for Remote Attestation is to measure a system's state by recording or evaluating software contents in storage and/or execution by the processor, communicating the evaluation to a interested party, and the comparison by the interested party of the received data against the established baseline. For example, TCG's "Integrity Management" process involves five phases:

> "**Creation**: Integrity metrics of components are generated by hardware manufacturers and software vendors, and the Reference Manifests for these components are issued.
> **Collection**: When components are gathered to constitute a computing platform, corresponding integrity information is collected. E.g., this step is done by the IT department of a corporation that is responsible for configuration of client PCs in the enterprise.
> **Communication/Publication**: Collected integrity information is made available to the verifier, e.g., through a public repository of RMs [Reference Manifests].
> **Collation**: When the verifier seeks to verify an integrity report, the verifier collects the RMs in order to use it for evaluation. This step is called collation.
> **Evaluation**: The verifier evaluates the integrity report against the RMs, as well as additional information (e.g., the latest vulnerability information) if such information is available."
> (Ex. 1005 at p. 4).

67. One of the challenges in Remote Attestation is knowing if the remote system is "lying." After all, what prevents malware from simply not reporting itself? One approach, which is used by TCG, is "a security module called the Trusted Platform Module (TPM), which is usually implemented as a tamper-resistant hardware module. In addition to serving as a cryptographic co-processor and protected storage for secrets and keys, the TPM is used to measure and report platform integrity in a manner that cannot be compromised even by the platform owners or the software running on it." (Ex. 1005 at p. 2).

68. Another characteristic of Remote Attestation is a "trusted bootstrap process." The basic idea is that it does no good to verify one software component if a previous software component (e.g., that the later component may rely on) has problems. The process must start at the beginning, meaning the lowest level code, such as BIOS boot code. Once that component is verified, each subsequent component is also verified, creating attestation of the entire set of software of the system. TCG's approach is described as:

> TCG defines the trusted bootstrap process [13], that comprises an iterative process of "measurement", loading, and execution of software components.

> When the system is powered-on, the immutable initial bootstrap code (such as the BIOS boot block) measures the next component and stores the measurement in the TPM before transferring control to the next component. subsequent steps, each software component recursively measures the next component and records the measurements in the TPM, until the operating system is loaded. The boot block and the TPM are called the Core Root of Trust for Measurement (CRTM), because they need to be trusted in order to trust the measurements in the trusted bootstrap.
> (Ex. 1005 at p. 2).

69. Once the data is collected, it is provided to the remote requester with the poofs. "Using TCG's [Trusted Computing Group] Remote Attestation, the integrity data for each component running on the platform is recorded in the security chip and securely checked by remote attestation." (Ex. 1005 at p. 1). The attestation proofs are protected by cryptography that prevents alteration or forgery. Again, using the TCG Remote Attestation as an example:

> In the attestation process, the TPM signs the PCR [Platform Configuration Register] values [which record the integrity measurements] and an external 160-bit challenge (such as a nonce from a challenger to avoid replay attacks) using an RSA private key, the confidentiality of which is protected by the TPM. The attestation is an atomic, protected operation and the attestation signature cannot be forged by malicious software. Therefore, if the TPM is properly designed and implemented to adhere to the TPM specifications, and the platform including the initial bootstrap code, is properly integrated with the TPM, a remote verifier can have confidence in the integrity measurements reported by the TPM.
> (Ex. 1005 at p. 3).

70. Finally, this digital signature is sent via communication channels to attest to the correct functionality and integrity of the system in question.

> Typically, remote attestation is used in a protocol where a requester (i.e., a computing platform with TPM) is seeking some interaction with a relying party (e.g., a service or an access to some resources), and the relying party needs to trust not only (or not necessarily) the identity of the requester but the integrity of the computing platform. The requester sends a platform integrity report to the relying party. The relying party may act as a verifier, or asks a trusted verifier to verify the report on its behalf. In either case, since the verifier may not be in the same trust domain as the requester, it may not know all of the software and hardware components that constitute

> the requesting computing platform. The Reference Manifest is used by the verifier as the authoritative information to evaluate the integrity report. (Ex. 1005 at p. 4).

VI. OVERVIEW OF THE CHALLENGED PATENT

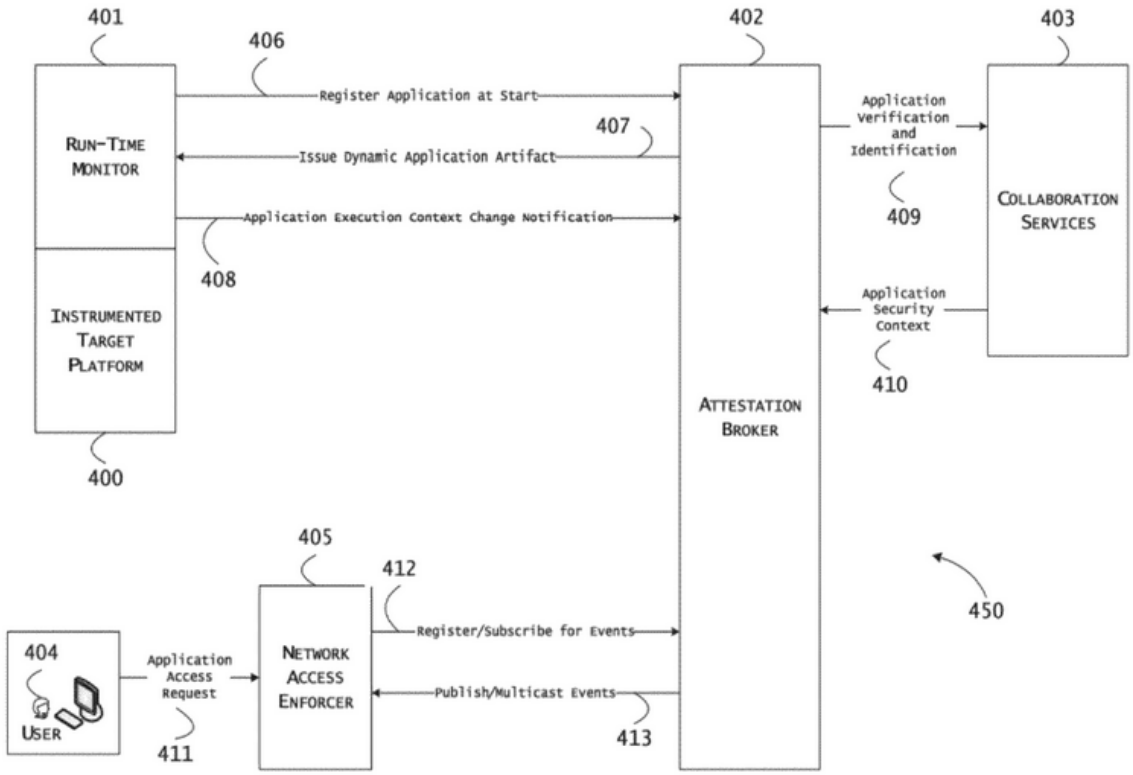
A. THE '441 PATENT SPECIFICATION AND CLAIMS

71. U.S. Patent No. 8,327,441 to Srinivas Kumar et al., titled "System and method for application attestation," issued on December 4, 2012. The '441 patent proposes a system by which to obtain trust of an application running on a remote system (hereafter described as a "instrumented target platform"). To obtain trust of the applications being evaluated '441 patent explains a method to perform attestation, authentication, and authorization of applications on behalf of a potential client of those applications (client henceforth is maintained as the user of the remote application running on the "instrumented target platform"). Finally, the '441 patent explains that a connection may be made between a potential client and a successfully attested application by disclosing a "method, apparatus and/or system to establish user-to-application connections based on the dynamic attestation of applications" (Ex. 1001 at 1:41-44). For example, without a system like this in place a user could mistakenly submit sensitive information to an application that may use that information maliciously. The '441 patent proposes a solution to this problem through remote attestation, wherein the application's functionality is locally evaluated via a runtime monitor, which is a separate trusted subsystem running on the "instrumented target platform."

72. The specification provides more details surrounding the attestation process. The '441 patent discloses a "runtime execution context", which comprises "executable file binaries and loaded components of the application", and a "security context", which comprises security information and an execution analysis for the runtime execution context. These contexts are sent to an attestation broker which generates a report indicating security risks and a security assertion is generated in association with this report and these risks.

73. Figure 4 below depicts the above explanation, wherein a user is attempting to access an application being provided as a remote application served from a "instrumented target platform." For the user to gain access to the application running on the "instrumented target platform," the "runtime monitor" must first produce the "runtime execution context" and "security context" to be later evaluated through the "attestation broker". This occurs in the "Register Application at Start" step in Figure 4. Figure 4 is referenced hereinbelow:

FIG. 4



(Ex. 1001 at Figure 4).

74. After the report has been generated, collaboration services authenticate the application (Ex. 1001 at Claim 6). If the application is not authenticated, the user's transaction

with the application served by the "instrumented target platform" will be controlled or restricted based on the results of the attestation.

75. After an attestation has been performed by the attestation broker, the attestation broker generates an artifact based on the contexts provided by the runtime monitor. As indicated by the "Application Execution Context Change Notification" stemming from the "runtime monitor", the subsequent consecutive attestations of an application being evaluated on behalf of a client may use a persisted artifact which recognizes changes in the application's execution context, comprised of executable file binaries and loaded components. The specification further explains, "The attestation service may be configured to generate an artifact based on a runtime local execution context of the running application instance on the instrumented target platform." (Ex. 1001 at 2:40-43).

76. The specification describes that this artifact may comprise a hash and further explains, "The attestation broker 109 may generate and may return an application artifact 107 for the running application 103 or 104 to the runtime monitor 102, as a reference for subsequent execution context change notifications 207 (see FIG. 2) for the applications… For example, when the hash of the targeted application changes or the file size of the targeted application changes, an updated attestation may be initiated by the runtime monitor 102 and provided by the attestation broker 109." (Ex. 1001 at 9:65-10:2).

77. Because the runtime monitor continually evaluates the applications running on a "instrumented target platform", an application on this platform can continually be trusted by users without need for additional attestations unless the executables that comprise the applications being run change.

B. PROSECUTION HISTORY

78. Prosecution of the 13/399,065 application that issued as the ’441 patent began with a non-final Office Action on April 27, 2012, in which claims 1-26 were rejected under 35 U.S.C. § 102(e) as unpatentable under U.S. Patent Publication 2011/0179477 to Starnes. (Ex. 1002 at pp. 111-119). Applicant initiated an Examiner Interview on June 7, 2012. (*Id.* at p. 107).

79. After an Examiner Interview, Taasera responded on June 29, 2012, by amending claims 1, 3-5, 13-19, and 21-26. Of relevance to the Challenged Claims, independent claim 1 was amended to recite “wherein the attributes comprise one or more executable file binaries of the application and loaded components of the application... wherein the security information comprises an execution analysis of the one or more executable file binaries and the loaded components”.

80. A Notice of Allowance then issued September 17, 2012, without an Examiner’s statement of reasons for allowance. (*Id.* at pp. 48-49).

VII. LEVEL OF SKILL IN THE ART

81. Based on my knowledge and experience, I understand that a person of ordinary skill in the art (“POSITA”) at the time of the earliest claimed priority date for the ’441 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. More education can supplement practical experience and vice versa.

82. I meet that definition of a person of ordinary skill in the art, and I did so on in and before February 2011. I also have worked with persons of ordinary skill in the art as part of my professional and academic experience, and I have an understanding of their skill level in and before February 2011.

VIII. OVERVIEW OF THE PRIOR ART

A. OVERVIEW OF "INTEGRITY MANAGEMENT FOR TRUSTED COMPUTING" BY MUNETOH, ET. Al ("MUNETOH")

83. The research paper "Integrity Management for Trusted Computing" by Seiji Munetoh, Megumi Nakamura, Sachiko Yoshihama, and Michiharu Kudo ("Munetoh") was published May 2008 in the Journal IEICE TRANSACTIONS on Information and Systems Vol.E91-D No.5 pp.1242-1251. Munetoh both provides an overview of TCG's trusted computing architecture as well as discloses additions tested in a prototype system developed by the authors.

84. The Trusted Computing Group (TCG) is an organization that concerns itself with the internal trust of a computer via a Trusted Platform Module (TPM). "The TCG [2] defines a set of industry-standard specifications for hardware and software to enable trusted computing among a wide variety of computing platforms, such as PCs, servers, mobile phones, etc." (Ex. 1005 at p. 2). The TCG has published well known and defined processes for which to evaluate a machine and provide two primary sources of information which include ensuring the genuineness of a system's TPM and the integrity of the system as measured by that TPM (Ex. 1005 at p. 3).

85. Munetoh explains that the TCG technology is used to ***remotely*** determine the "health" of a system.

> The TCG's trusted computing technology makes it possible to ***check the health of a computing platform (using a technique called Remote Attestation***) in a more secure and trusted manner and to alert the user whenever the platform is contaminated by unknown software. Using TCG's Remote Attestation, the integrity data for each component running on the platform is recorded in the security chip and securely checked by remote attestation.
> (Ex. 1005 at p. 1, emphasis added).

86. Munetoh explains that the attestation service enabled parties with a TPM to prove to a "relying party" both their identity and the integrity of their computing platform.

> Typically, ***remote attestation is used in a protocol where a requester (i.e., a computing platform with TPM) is seeking some interaction with a relying party (e.g., a service or an access to some resources), and the relying party needs to trust not only (or not necessarily) the identity of the requester but the integrity of the computing platform.*** The requester sends a platform integrity report to the relying party. The relying party may act as a verifier, or asks a trusted verifier to verify the report on its behalf. In either case, since the verifier may not be in the same trust domain as the requester, it may not know all of the software and hardware components that constitute the requesting computing platform. The Reference Manifest is used by the verifier as the authoritative information to evaluate the integrity report. (Ex. 1005 at p. 4, emphasis added).

87. Munetoh further explains that the measurement of the computing platform includes applications: "The PTS [Platform Trust Services] allows applications running on the platform to be measured and attested before it starts using the service, and thus allowing to establish trust between the client and the server." (Ex. 1005 at p. 5). This is also visualized in Munetoh's Figure 1, reproduced herein below:

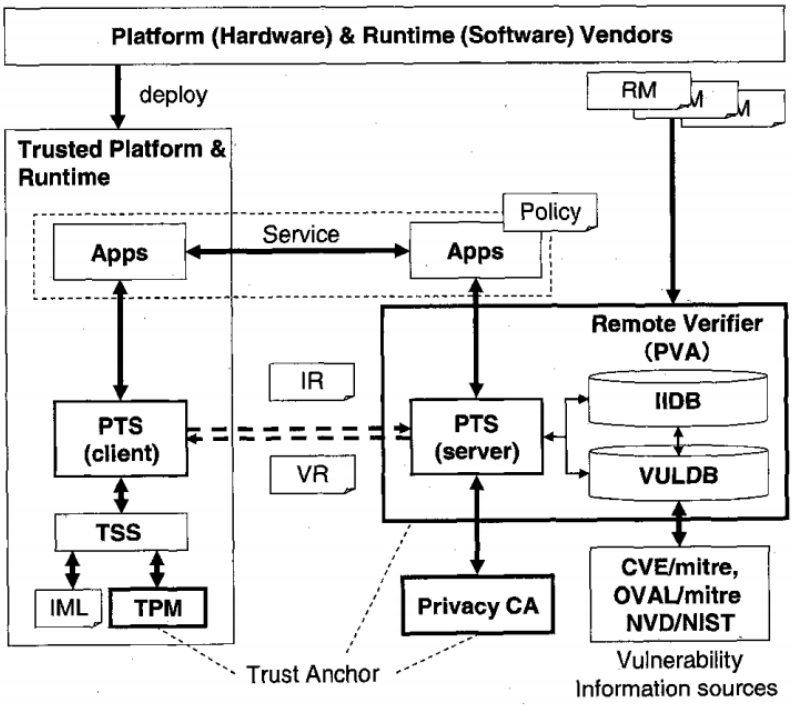


Fig. 1 PTS Client/Server architecture.

(Ex. 1005 at Figure 1).

88. In this Figure 1 from Munetoh, the PTS client transmits an Integrity Report (IR) to the server, which sends back a Verification Response (VR). This enables the server's apps to trust the client's apps (assuming the Verification Result is successful).

89. Munetoh explains that trusting the integrity of a running application involves first performing an integrity measurement of the "bootstrap", meaning all of the other software necessary for the application to run. This primarily includes the operating system and other low level code (e.g., BIOS boot code). Munetoh explains that this element constitutes a "trusted bootstrap":

> Platform integrity measurement consists of multiple phases of measuring and storing the fingerprints of the hardware and software components that make up a platform. Integrity measurement can be categorized into two types: ***1) the TCG trusted bootstrap and 2) other integrity measurements built on top of the trusted bootstrap***.
> (Ex. 1005 at p. 2, emphasis added).
>
> TCG defines the trusted bootstrap process [13], ***that comprises an iterative process of "measurement", loading, and execution of software components. When the system is powered-on, the immutable initial bootstrap code (such as the BIOS boot block) measures the next component and stores the measurement in the TPM before transferring control to the next component. In subsequent steps, each software component recursively measures the next component and records the measurements in the TPM, until the operating system is loaded***.
> (Ex. 1005 at p. 2, emphasis added).

90. Once the trusted bootstrap is loaded, it is possible to subsequently create a chain of trust to other running software.

> Integrity measurement can be categorized into two types: 1) the TCG trusted bootstrap and 2) ***other integrity measurements built on top of the trusted bootstrap***.
> (Ex. 1005 at p. 2, emphasis added).
>
> The Integrity Measurement Architecture [14] supports such measurements for the Linux kernel, and enhances the role of the TPM for measuring not only the static state of a system but also the dynamic state. Several proposals have been made [9], [10], [12] to measure additional states, including the

> configuration of the OS, middleware, and applications, to further enhance the role of TPM to the upper layers of the software stack. ***Care needs to be taken, though, that a transitive trust chain is maintained from the CRTM all the way to the component being measured.***
> (Ex. 1005 at p. 3, emphasis added).

91. In their own prototype, Munetoh proved the integrity of all components including running software, which are referred to as "user land components."

> We used a commercially available PC with a TPM chip and KNOPPIX 5.1.1 for Trusted Computing Geeks [27] as the target platform… This modified KNOPPIX supports the trusted boot, including the transitive trust chain which covers the ***CRTM, BIOS, bootloader, operating system, and the user land executables.*** The Integrity of the user land executables and libraries are measured with the Linux-IMA. ***Thus, the IR generated by the client-side PTS contains all of the components that are running or ran on the target platform.***
> (Ex. 1005 at p. 8).

92. Munetoh also discloses the use of an attestation server for providing the remote attestation services to parties requesting verification. As also explained above, the attestation server may be the relying party, meaning it will perform the attestation in order to determine if it, the attestation server, will interact with the requester after deeming it to be safe. Alternatively, it may be a third-party verifier providing the attestation result to other relying parties.

> Typically, remote attestation is used in a protocol where a requester (i.e., a computing platform with TPM) is seeking some interaction with a relying party (e.g., a service or an access to some resources), and the relying party needs to trust not only (or not necessarily) the identity of the requester but the integrity of the computing platform. The requester sends a platform integrity report to the relying party. ***The relying party may act as a verifier, or asks a trusted verifier to verify the report on its behalf. In either case, since the verifier may not be in the same trust domain as the requester, it may not know all of the software and hardware components that constitute the requesting computing platform.*** The Reference Manifest is used by the verifier as the authoritative information to evaluate the integrity report.
> (Ex. 1005 at p. 4, emphasis added).

93. Munetoh also discloses an exemplary architecture for the various components discussed above. In said architecture, the attestation server is described as the "Remote Verifier" or the "Platform Verification Authority", which contains a "server-side PTS" as a component. This server trusts an application on the client, which is described as a "trusted platform" containing a "client-side PTS", based on the attestation data sent by the client. The server contains various components used for verification including a vulnerabilities database. This architecture is visualized in Munetoh's Figure 3, which is reproduced hereinbelow:

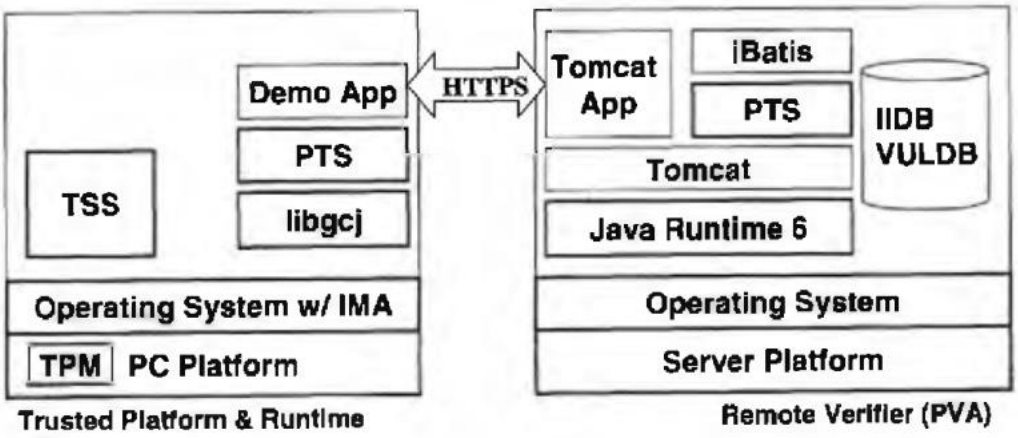


Fig. 3 Our prototype implementation of PTS and demo app.

(Ex. 1005 at Figure 3).

B. OVERVIEW OF "TISA: TOWARD TRUSTWORTHY SERVICES IN A SERVICE-ORIENTED ARCHITECTURE" TO RAJAN

94. The research paper "Tisa: Toward Trustworthy Services in a Service-Oriented Architecture" by Hridesh Rajan and Mahantesh Hosamani was published on December 18, 2008 in the Journal "IEEE Transactions on Services Computing," Vol. 1, No. 4.

95. Rajan discloses a system wherein remote attestation, like the remote attestation described in Munetoh, enables trust in a service-oriented architecture. Rajan explains, "SERVICE-ORIENTED Computing (SoC) is a new programming paradigm that promotes dealing with

modules or components, often called services [1]… are often deployed on servers that are not owned or operated by the clients." (Ex. 1006 at p. 1). Rajan discloses providing an attestation broker that provides attestation results for the servers providing services to the client. Figure 1 of Rajan illustrates how the three parties (i.e., Broker, Client, and Provider):

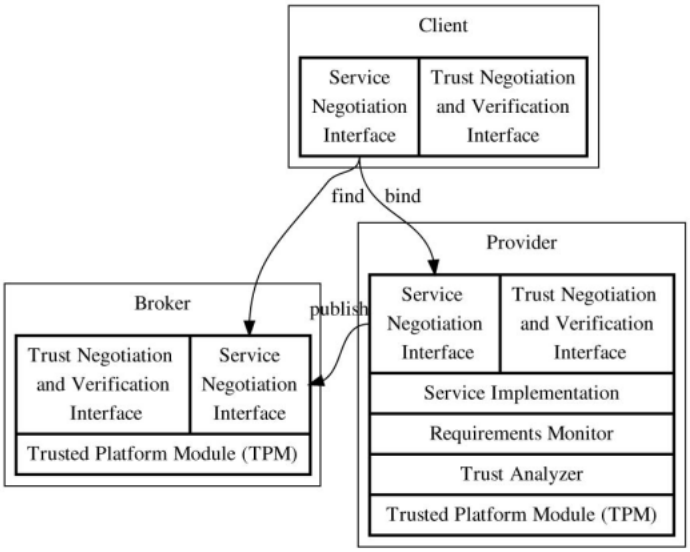


Fig. 1. Proposed architectural extensions.

(Ex. 1006 at Figure 1).

96. A "Provider" provides a service to the "Client." There may be many Providers that provide the same service; a "Broker" enables the Client to connect with an appropriate Provider of a service. To this existing architecture, Rajan enables the Clients to be able to trust Providers through the Broker.

97. As shown in Rajan's Figure 2, of which an annotated version is reproduced hereinbelow, of Rajan that shows the Provider of the service sending integrity measurements for attestation to the broker.

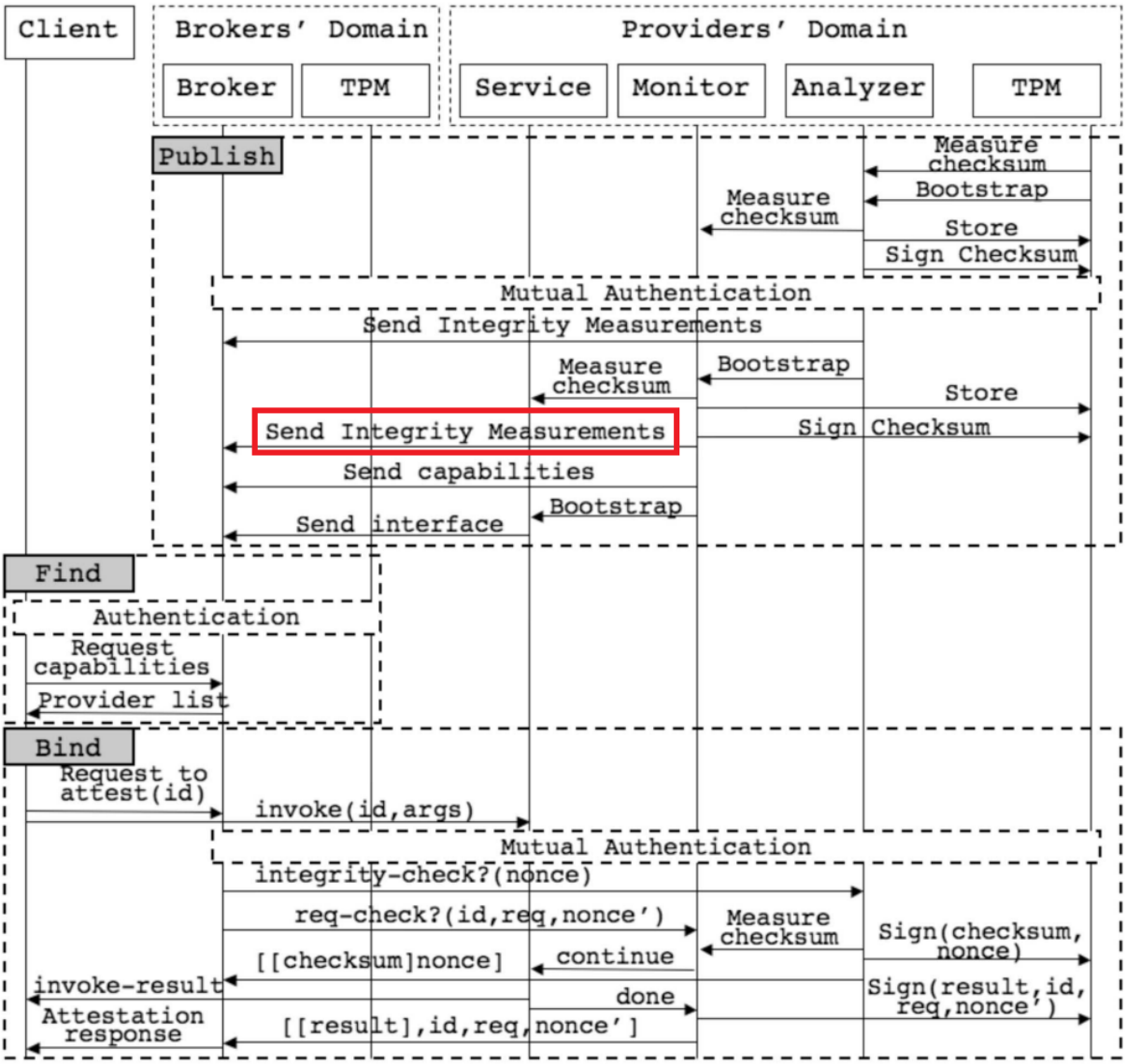


Fig. 2. Overview of Interaction between Components.

(Ex. 1006 at Figure 2).

98. In Rajan, service providers register with the broker and identify themselves by the functionality (services) and the attestation capabilities they provide:

> Traditional interaction pattern publish-find-bind between components in SoAs is augmented with steps for negotiating requirements monitoring capabilities as shown in Fig. 2. ***During the publish step, in addition to the functional properties, service providers also make the capabilities of the requirements monitor available to the broker via the trust negotiation interface. By capabilities, we mean the type of nonfunctional***

> ***requirements that the monitor can verify.*** For example, a service provider that deploys a monitor similar to Barbon et al.'s [5] work would make the capabilities to verify Boolean, statistical, and timing properties available.
> (Ex. 1006 at p. 3, emphasis added).

99. When a service client requests a service, the broker identifies services that both provide the necessary functionality as well as the necessary attestation. As Rajan explains, "[t]he broker responds to [the service client's] request with a list of only those services that satisfy both functional requirements as well as provide monitoring capabilities for nonfunctional requirements." (Ex. 1006 at p. 4). Rajan explains that the same kind of attestation operations (e.g., recording integrity measurements in PCR registers) disclosed by Munetoh are used for proving that the service is compliant:

> During the publish step, the trust analyzer and requirements monitor send integrity measurements to the broker after mutual authentication. ***The integrity measurements are stored in the platform configuration registers (PCRs) of the TPM using the SHA1 hashing algorithm***. The key idea is to store integrity measurements as successive hash values. For example, let us assume that integrity measurements of the trust analyzer resulted in a hash value of $hash_{TA}$ in the PCR. The integrity measurement for the requirements monitor now uses $hash_{TA}$ as its base value.
> (Ex. 1006 at p. 4, italicized $hash_{TA}$ original, other emphases added).

100. Rajan discloses how the attestation capabilities are used for monitoring specific programs (or applications) while they are running.

> In this work, we focus on the trace-based requirements monitor… In a trace-based monitor, requirements are specified as logical formulas ***over program traces***. An example of such a trace-based requirement for the payment processing service could be, "***in no trace of the service implementation, the argument credit card number directly or indirectly flows to a persistence-related method, e.g., File.write(...)***."
> (Ex. 1006 at pp. 5-6, emphasis added).

101. Rajan, like Munetoh, uses the TCG architecture rooted in a TPM to provide the integrity measurements used for attestation. As Rajan explains:

> Over the past few years, a consortium of key industry players under the umbrella of Trusted Computing Group(TCG) [19] have developed the specification for the trusted platform module (TPM) [20] with the goal to guarantee security, integrity, and confidentiality of data through innovative hardware-based architectures. Both TCG and alternatives [21] aim to bootstrap higher level trust from rudimentary TPM supported trust using some software trust architecture or design principle.
>
> A TPM is a trusted agent coprocessor within a remote computing platform which derives its root of trust from its manufacturer or a delegated trusted third party [19]. A TPM can be trusted to perform certain actions truthfully, despite being an integral part of a potentially malicious or compromised system. In other words, it is our trusted ambassador in a friendly or hostile foreign territory. The TPM hardware, firmware, and software provide a root of trust. A TPM can extend its trust to higher layers of the system by building a chain of trust starting from the hardware and subsequently linking upper layers.
> (Ex. 1006 at p. 2).

102. The integrity measurements provided by Rajan include two components. First, there is an attestation of the "software stack" which includes all of the low-level components, plus a "requirements monitor."

> During service execution, the broker and the provider must communicate to verify that: ***the provider and the broker are indeed the same principals that agreed to the contract during the publish step, the requirements monitor as well as the software stack on the provider's end is not compromised***, and the trace provided by this monitor does not violate the properties desired by the client.
>
> The trust analyzer component, which derives its root of trust from the TPM on the provider's side, serves to verify the integrity of the requirements monitor. ***It provides a report that includes an immutable integrity measurement of the local architecture along with a time stamp to indicate the freshness of the measurement***.
> (Ex. 1006 at p. 6, emphasis added)
>
> This trace, along with static checksum of the software stack, is sent to the trusted third party [broker] (step 7). ***To remind the reader, the static checksum is created by computing the SHA1 hash of the software stack up to the requirements monitor*** and is essential to validate the current state of the service implementation (step 5). This hash is signed using the AIK on the TPM of the service provider (step 6).
> (Ex. 1006 at p. 7, emphasis added).

103. The requirements monitor, now trusted by attestation, is then used to analyze the performance of the service software at runtime. Rajan discloses a "trace-based" attestation to perform the attestation of the run-time behavior of the software.

> In this work, we focus on the trace-based requirements monitor… In a trace-based monitor, requirements are specified as logical formulas ***over program traces***. An example of such a trace-based requirement for the payment processing service could be, "***in no trace of the service implementation, the argument credit card number directly or indirectly flows to a persistence-related method, e.g., File.write(...)***."
> (Ex. 1006 at pp. 5-6, emphasis added).

104. Rajan discloses sending the validation result of both the software stack and any trace-based validations to the service client:

> During the service, the trust analyzer [of the service] monitors and creates a trace of the execution containing the field and method information identified during the initialization phase. The authentication server is responsible for verifying this trace. To that end, after receiving the signed data from the provider… if the data from the provider and its signature do not match, it represents corruption of the provider's information and, hence, a violation of trust… ***The client is notified of such violations (steps 6, 10)***.
>
> Upon successfully verifying the data and the nonce, the next step is to verify the trace (step 12). The traces from two different executions may differ due to modifications of service implementation as well as differences in input values. Thus, exact equivalence may not be used to report violations. Instead, we use the trace received from the provider to check if it contains a sequence that violates the requirements. In our example, persistence and serialization of data are assumed to be a violation of requirements; ***thus, the presence of an API call that persists or serializes data will be reported as violation…***
>
> Finally, if all checks pass, the system configuration is checked. If any of these checks fail, the clients are notified of a violation; ***otherwise, an assurance is sent to the client***.
> (Ex. 1006 at p. 7, emphasis added).

105. Rajan also discloses controlling a user's transaction (through the service client) with the application by applying a set of authorization rules in accordance with the attestation

result. The service client limits the user's transaction (i.e., with the service) by aborting the operation if a violation is reported. See, for example, Procedure 5 from Rajan:

```
Procedure 5. Bind step for client's end.
 1. initTransaction();
 2. requestCert(brokerAddr,providerAddr,Id);
 3. sendRequest(providerAddr, params);
 4. result = receiveCert(brokerAddr,Id);
 5. if result == violationNotification OR timeOut then
 6.   abortTransaction();
 7. else
 8.   commitTransaction();
 9. end if
10. return;
```

(Ex. 1006 at p. 8).

IX. GROUND 1: CLAIMS 1-7 AND 9 ARE RENDERED OBVIOUS BY MUNETOH IN VIEW OF RAJAN

106. Claims 1-7 and 9 of the '441 patent are rendered obvious to a POSITA by Munetoh in view of Rajan, as explained in the sections below. I also explain that it would have been obvious for a POSITA to combine Munetoh and Rajan.

A. Motivation to Combine Munetoh and Rajan

107. A POSITA would have been motivated to combine Munetoh and Rajan. As discussed in Sections VIII.A. and VIII.B., Munetoh and Rajan are both based on Remote Attestation using TCG's specification. The core attestation technology, such as the use of TPMs and PCRs, is the same.

108. As also explained above, Munetoh, however, contributes elements for performing some of the steps of attestation that are not well defined by TCG's specification. Munetoh explains that "Our contribution for solving open issues in platform integrity validation includes 1) a state-machine based model for validation of integrity reports which enables verification of the transitive

trust chain of a platform runtime, 2) the integrity database which integrates the expected integrity metrics of software packages with vulnerability information obtained from external security advisories, 3) identified a set of vocabularies for expressing required properties of target platforms." (Ex. 1005 at pp. 5-6).

109. Rajan also builds an attestation system around a TPM architecture for "platform authentication." Rajan explains that the "Trusted Computing Group (TCG) [19] [has] developed the specification for the trusted platform module (TPM) [20] with the goal to guarantee security, integrity, and confidentiality of data through… hardware-based architectures." (Ex. 1006 at p. 2). Accordingly, the core TPM technology of Rajan is the same as the TPM technology disclosed in Munetoh.

110. Thus, a POSITA would have understood that the core technology for the two systems was compatible with each other. A POSITA also would have found it beneficial to incorporate some of the advantages of Rajan into the existing Munetoh disclosure.

111. To illustrate the improvements, I first note that Munetoh discloses an attestation client (called a Trusted Platform) and an attestation server (called a Platform Validation Authority, or PVA) as illustrated in Figure 1, reproduced hereinbelow:

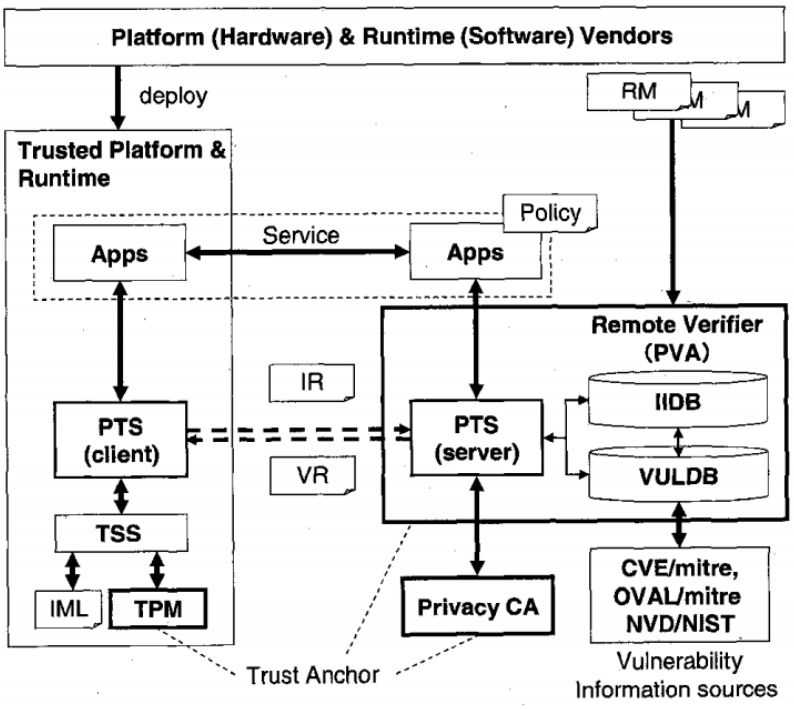


Fig. 1 PTS Client/Server architecture.

(Ex. 1005 at Figure 1).

112. A POSTIA would have been motivated to draw on Rajan, however, in order to deploy a broader system for interacting components. As Rajan explains, "SERVICE-ORIENTED Computing (SoC) is a new programming paradigm that promotes dealing with modules or components, often called services [1]… ***Among other goals, service-oriented computing promotes abstraction, loose coupling, and interoperability among services*** [2]." (Ex. 1006 at p. 1, emphasis added). As shown in Rajan's Figure 1, reproduced hereinbelow, Rajan discloses a more flexible architecture of three parties:

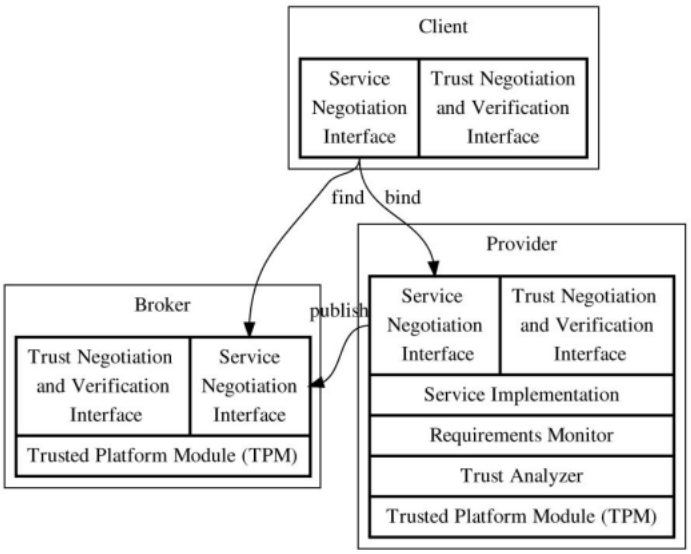


Fig. 1. Proposed architectural extensions.

(Ex. 1006 at Figure 1).

113. Rajan's Broker acts as an attestation server to the Providers. The Providers attest to the broker about their state, which the Broker uses on behalf of the Client.

114. Figure 2 of Rajan, an annotated version of which is reproduced hereinbelow, shows the Provider of the service (e.g., provided by a server) as sending the integrity measurements to the broker, just as the trusted platform send to the PVA in Munetoh.

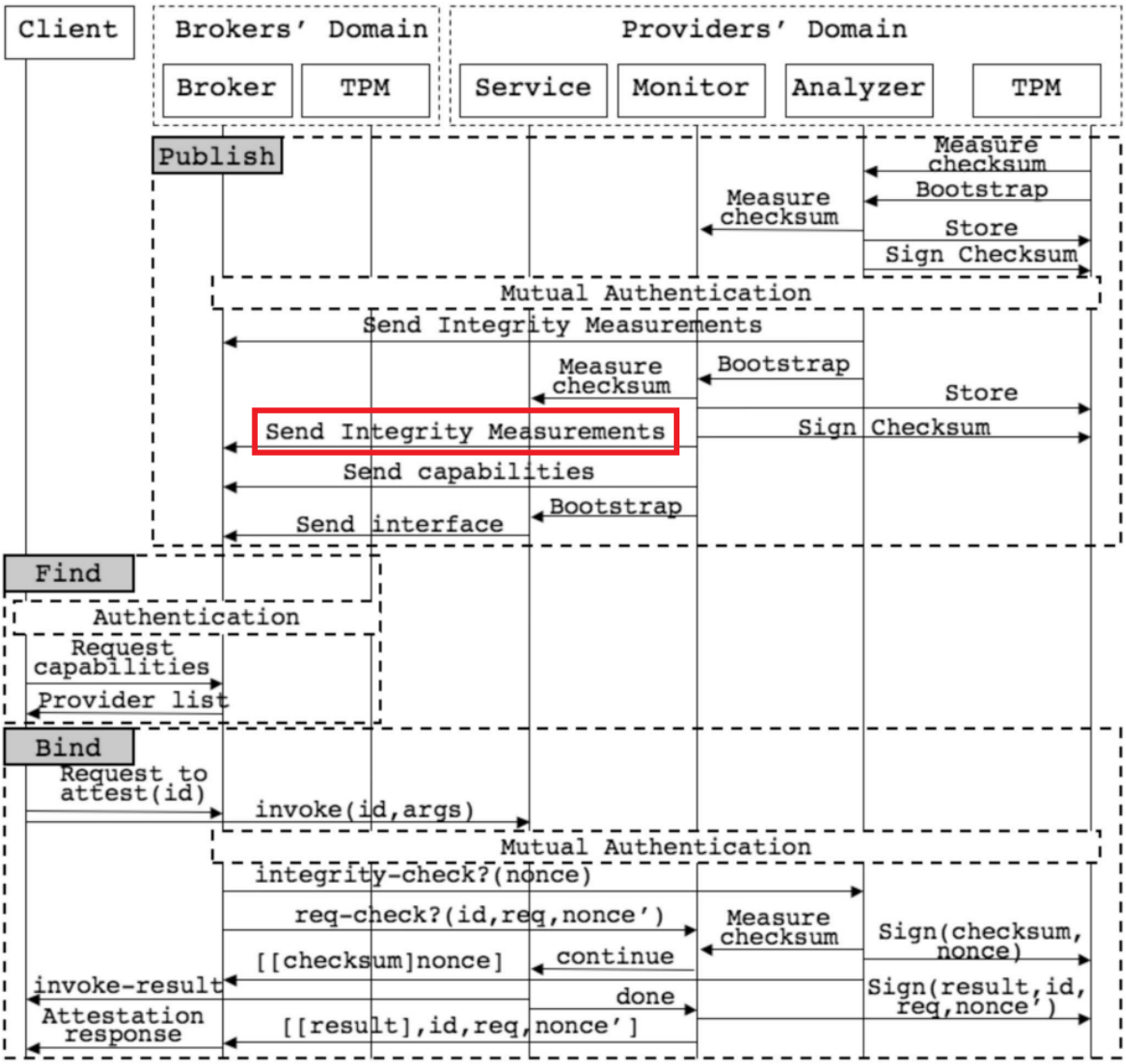


Fig. 2. Overview of Interaction between Components.

(Ex. 1006 at Figure 2, annotated).

115. A POSITA would have found Rajan's broker, with all of the advantages of the services architecture discussed above, to fit naturally into Munetoh. Munetoh explicitly discloses the concept of a "dedicated verifier." As Munetoh explains, "[t]he relying party may act as a verifier, or asks a trusted verifier to verify the report on its behalf." (Ex. 1005 at p. 4). Rajan's broker serves as a "trusted verifier" for the Client, which verifies the integrity reports (or integrity measurements) of the Provider on the Client's behalf.

116. Munetoh explains that the general goal of attestation is "a protocol where… the relying party needs to trust not only (or not necessarily) the identity of the requester but the integrity of the computing platform." (Ex. 1005 at p. 4). A POSITA would have looked to a reference like Rajan for how to implement these goals. Rajan, for example, discloses that service providers register with the broker and identify themselves by the functionality (services) and the attestation capabilities they provide:

> Traditional interaction pattern publish-find-bind between components in SoAs is augmented with steps for negotiating requirements monitoring capabilities as shown in Fig. 2. ***During the publish step, in addition to the functional properties, service providers also make the capabilities of the requirements monitor available to the broker via the trust negotiation interface. By capabilities, we mean the type of nonfunctional requirements that the monitor can verify.*** For example, a service provider that deploys a monitor similar to Barbon et al.'s [5] work would make the capabilities to verify Boolean, statistical, and timing properties available. (Ex. 1006 at p. 3, emphasis added).

117. When a service client requests a service, the broker identifies services that both provide the necessary functionality as well as the necessary attestation. As Rajan explains, "[t]he broker responds to [the service client's] request with a list of only those services that satisfy both functional requirements as well as provide monitoring capabilities for nonfunctional requirements." (Ex. 1006 at p. 4). In this way, a client of a service can engage with any service that provides both the necessary functionality as well as the necessary safety and integrity.

118. A POSITA also would have been motivated to incorporate Rajan's disclosures measuring runtime integrity of applications. Munetoh already discloses providing an integrity measurement of running binaries starting with the lowest level BIOS data and all subsequently loaded components (i.e., software). Because these components are loaded (i.e., running), the measurement is of one or more applications at runtime:

> We used a commercially available PC with a TPM chip and KNOPPIX 5.1.1 for Trusted Computing Geeks [27] as the target platform… This modified KNOPPIX supports the trusted boot, including the transitive trust chain which covers the ***CRTM, BIOS, bootloader, operating system, and the user land executables.*** The Integrity of the user land executables and libraries are measured with the Linux-IMA. ***Thus, the IR generated by the client-side PTS contains all of the components that are running or ran on the target platform.***
> (Ex. 1005 at p. 8).

119. However, as discussed in Section VIII.B., Rajan discloses a more fine-grained analysis of runtime behavior of application by tracing their behavior with a requirements monitor. A POSITA would have been motivated to use Rajan's approach for "trace-based" attestation to attest to the run-time behavior of the software.

> In this work, we focus on the trace-based requirements monitor… In a trace-based monitor, requirements are specified as logical formulas ***over program traces***. An example of such a trace-based requirement for the payment processing service could be, "***in no trace of the service implementation, the argument credit card number directly or indirectly flows to a persistence-related method, e.g., File.write(...)***."
> (Ex. 1006 at pp. 5-6, emphasis added).

120. A POSITA would have understood that combining Rajan's trace-based analysis with Munetoh would enable attestation of the composition of the software at runtime as well as precise behavior (such as writing files). This would provide better runtime analysis and better security of the system.

121. Thus, a POSITA would have been motivated to combine Rajan's advantages and capabilities with Munetoh. And because of the similarity in core technologies (e.g., the TCG's TCB-based attestation), a POSITA would have required little experimentation and had a high expectation of success.

B. Claim 1 is rendered obvious by Munetoh in view of Rajan

122. Each of the limitations of Claim 1 of the '441 patent is rendered obvious to a POSITA by Munetoh in view of Rajan as described below.

1. Limitation 1[a]

123. Limitation 1[a] of the '441 patent reads:

A method of providing an attestation service for an application at runtime executing on a computing platform using an attestation server, comprising:

124. I understand that limitation 1[a] is a "preamble" and that the preamble of claims is often not limiting. To the extent that limitation 1[a] is limiting, Munetoh and Rajan each discloses it.

125. Munetoh discloses a method of providing an attestation service for an application at runtime executing on a computing platform. Munetoh explains that such methods were common and a reference architecture was defined by the Trusted Computing Group. The well-known concept was called "Remote Attestation". As explained in Section V.A.,

> The TCG's trusted computing technology makes it possible to ***check the health of a computing platform (using a technique called Remote Attestation***) in a more secure and trusted manner and to alert the user whenever the platform is contaminated by unknown software. Using TCG's Remote Attestation, the integrity data for each component running on the platform is recorded in the security chip and securely checked by remote attestation.
> (Ex. 1005 at p. 1, emphasis added).

126. Munetoh explains that the attestation service enabled parties with a TPM to prove to a "relying party" both their identity and the integrity of their computing platform.

> Typically, ***remote attestation is used in a protocol where a requester (i.e., a computing platform with TPM) is seeking some interaction with a relying party (e.g., a service or an access to some resources), and the relying party needs to trust not only (or not necessarily) the identity of the requester but the integrity of the computing platform.*** The requester sends a platform integrity report to the relying party. The relying party may act as a verifier, or asks a trusted verifier to verify the report on its behalf. In either case, since the verifier may not be in the same trust domain as the requester, it may not know all of the software and hardware components that constitute the requesting computing platform. The Reference Manifest is used by the verifier as the authoritative information to evaluate the integrity report.

(Ex. 1005 at p. 4, emphasis added).

127. Munetoh further explains that the measurement of the computing platform includes applications: "The PTS [Platform Trust Services] allows applications running on the platform to be measured and attested before it starts using the service, and thus allowing to establish trust between the client and the server." (Ex. 1005 at p. 5). This is also visualized in Munetoh's Figure 1, reproduced and annotated hereinbelow:

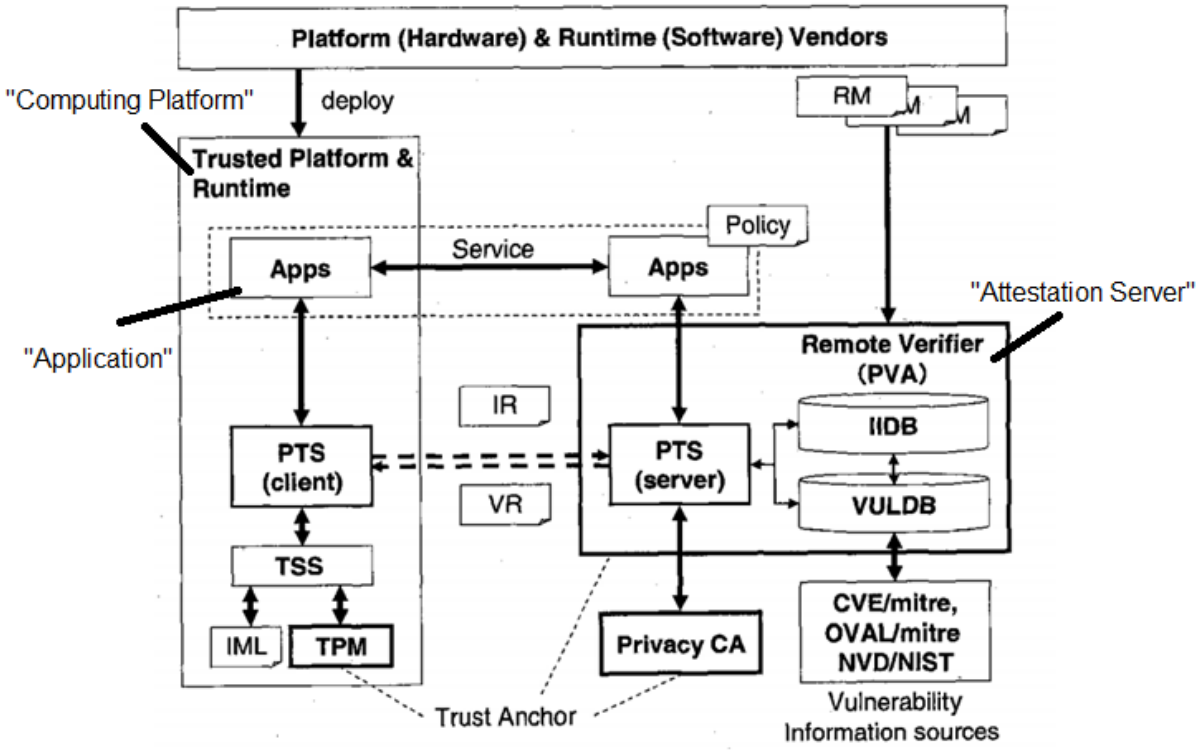


Fig. 1 PTS Client/Server architecture.

(Ex. 1005 at Figure 1, annotated).

128. In this Figure 1 from Munetoh, the PTS client transmits an Integrity Report (IR) to the server, which sends back a Verification Response (VR). This enables the server's apps to trust the client's apps (assuming the Verification Result is successful).

129. Munetoh explains that the integrity measurements are used for attesting to the integrity of running software (i.e., "applications"). First, the integrity measurements cover the lower-level software including the operating system.

> Integrity measurement can be categorized into two types: 1) the TCG trusted bootstrap and 2) ***other integrity measurements built on top of the trusted bootstrap***.
> (Ex. 1005 at p. 2, emphasis added).

> The Integrity Measurement Architecture [14] supports such measurements for the Linux kernel, and enhances the role of the TPM for measuring not only the static state of a system but also the dynamic state. Several proposals have been made [9], [10], [12] to measure additional states, including the configuration of the OS, middleware, and applications, to further enhance the role of TPM to the upper layers of the software stack. ***Care needs to be taken, though, that a transitive trust chain is maintained from the CRTM all the way to the component being measured.***
> (Ex. 1005 at p. 3, emphasis added).

130. In their own prototype, Munetoh proved the integrity of all components including running software, which is referred to as "user land components."

> We used a commercially available PC with a TPM chip and KNOPPIX 5.1.1 for Trusted Computing Geeks [27] as the target platform… This modified KNOPPIX supports the trusted boot, including the transitive trust chain which covers the ***CRTM, BIOS, bootloader, operating system, and the user land executables.*** The Integrity of the user land executables and libraries are measured with the Linux-IMA. ***Thus, the IR generated by the client-side PTS contains all of the components that are running or ran on the target platform.***
> (Ex. 1005 at p. 8).

131. Munetoh also discloses the use of an attestation server for providing the remote attestation services to parties requesting verification. As also explained above, the attestation server may be the relying party, meaning it will perform the attestation in order to determine if it, the attestation server, will interact with the requester after deeming it to be safe. Alternatively, it may be a third-party verifier providing the attestation result to other relying parties.

> Typically, remote attestation is used in a protocol where a requester (i.e., a computing platform with TPM) is seeking some interaction with a relying party (e.g., a service or an access to some resources), and the relying party needs to trust not only (or not necessarily) the identity of the requester but the integrity of the computing platform. The requester sends a platform integrity report to the relying party. ***The relying party may act as a verifier, or asks a trusted verifier to verify the report on its behalf. In either case, since the verifier may not be in the same trust domain as the requester, it may not know all of the software and hardware components that constitute the requesting computing platform.*** The Reference Manifest is used by the verifier as the authoritative information to evaluate the integrity report.
> (Ex. 1005 at p. 4, emphasis added).

132. Munetoh also discloses an exemplary architecture for the various components discussed above. In said architecture, the attestation server is described as the "Remote Verifier" or the "Platform Verification Authority", which contains a "server-side PTS" as a component. This server trusts an application on the client, which is described as a "trusted platform" containing a "client-side PTS", based on the attestation data sent by the client. The server contains various components used for verification including a vulnerabilities database. This architecture is visualized in Munetoh's Figure 3, which is reproduced hereinbelow:

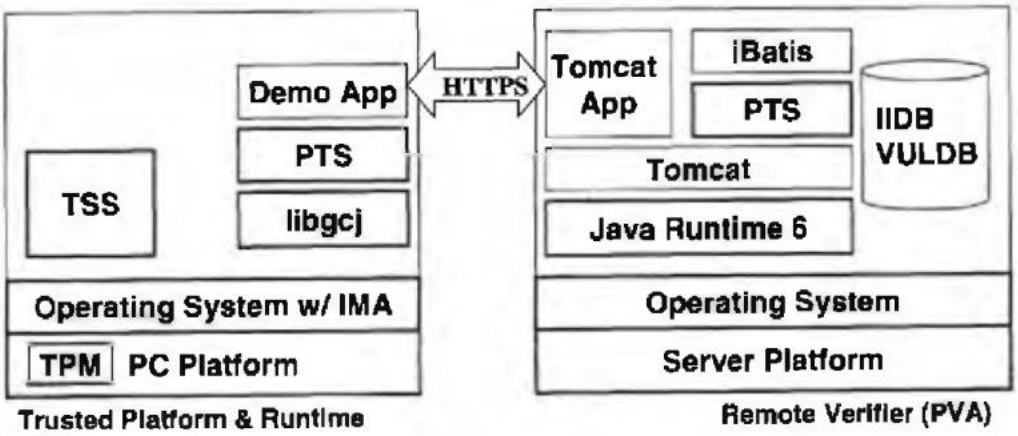


Fig. 3 Our prototype implementation of PTS and demo app.

(Ex. 1005 at Figure 3).

133. As explained in Section IX.A., a POSITA would have been motivated to combine Rajan with Munetoh in order to take advantage of the more flexible system architecture and the fine-grained attestation of runtime behavior.

134. However, before explaining Rajan's combination for this limitation, I need to clarify the terms "client" and "server." These terms are used to refer to different elements depending on context and clarification is required so as the terms are not confused between Munetoh and Rajan. I note that the term "client" and "server" are always relative to the perspective of the service being provided. That is a "server" provides a service to a "client." The client may offer its own services to other systems, in which case it is a server to those other systems. In Munetoh, the attestation client (trusted platform) transmits a request for verification to the attestation server, as described above. The attestation client may, itself, be a server for some other kind of service. In other words, in Munetoh, the terms "client" and "server" refer to the attestation service.

135. Rajan's use of "client" and "server" do not refer to the attestation server. Instead, client and server in Rajan's disclosures refer to some other kind of service. As Rajan explains, "SERVICE-ORIENTED Computing (SoC) is a new programming paradigm that promotes dealing with modules or components, often called services [1]… are often deployed on servers that are not owned or operated by the clients." (Ex. 1006 at p. 1). Rajan discloses providing an attestation broker that provides attestation results for the servers providing services to the client. However, the server disclosed by Rajan that provides the services is, from the perspective of Munetoh, the equivalent of the trusted platform (or attestation client) requesting attestation from a PVA or remote verifier.

136. This is clearly shown in Figure 2, of which an annotated version is reproduced hereinbelow, of Rajan that shows the "Provider" of the service (e.g., provided by a server) as sending the integrity measurements to the broker.

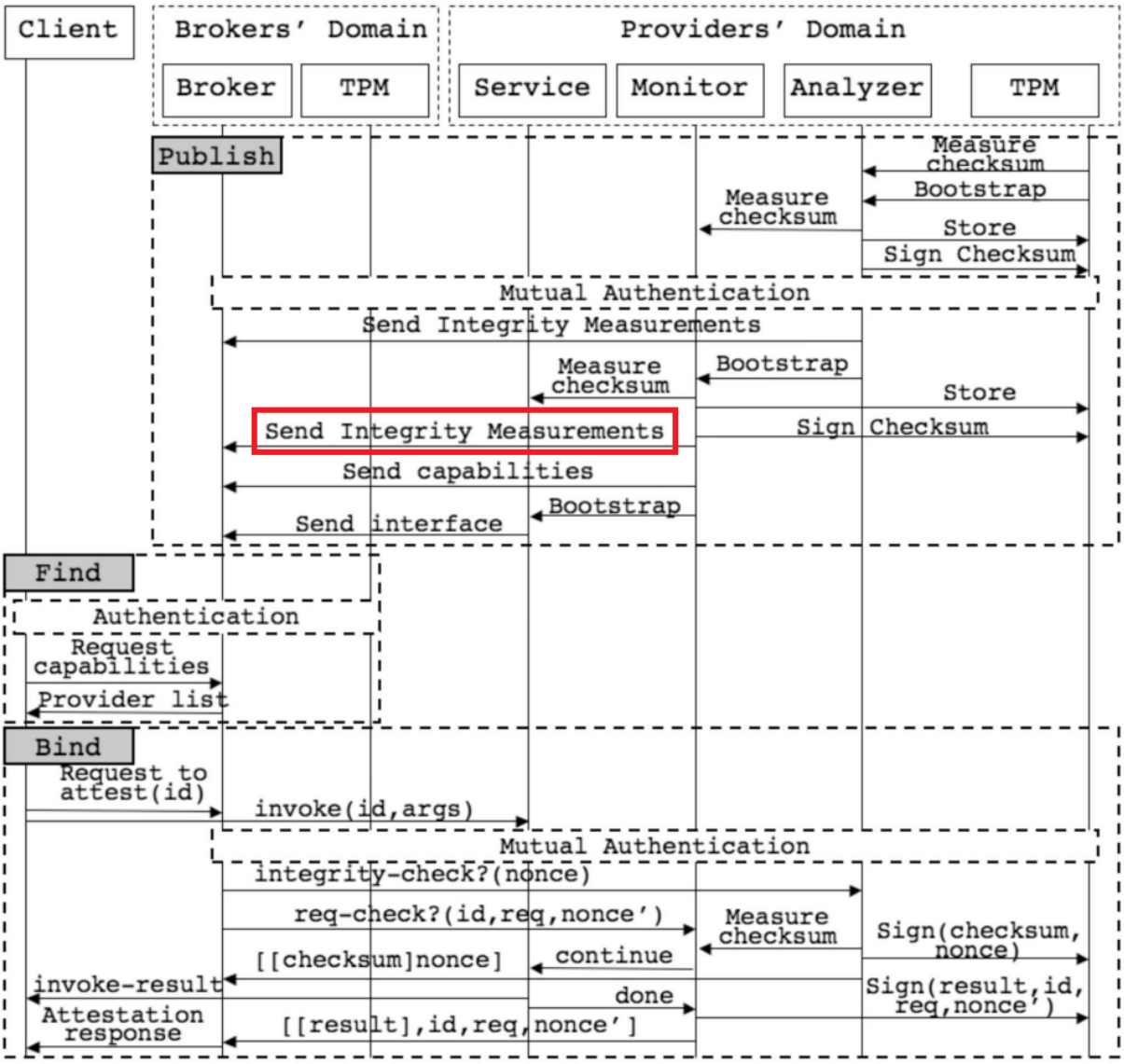


Fig. 2. Overview of Interaction between Components.

(Ex. 1006 at Figure 2, annotated).

137. This makes the service provider equivalent to the client-side PTS and the broker equivalent to the server-side PTS in the Munetoh Fig. 1, which though already cited above is reproduced hereinbelow for clarity:

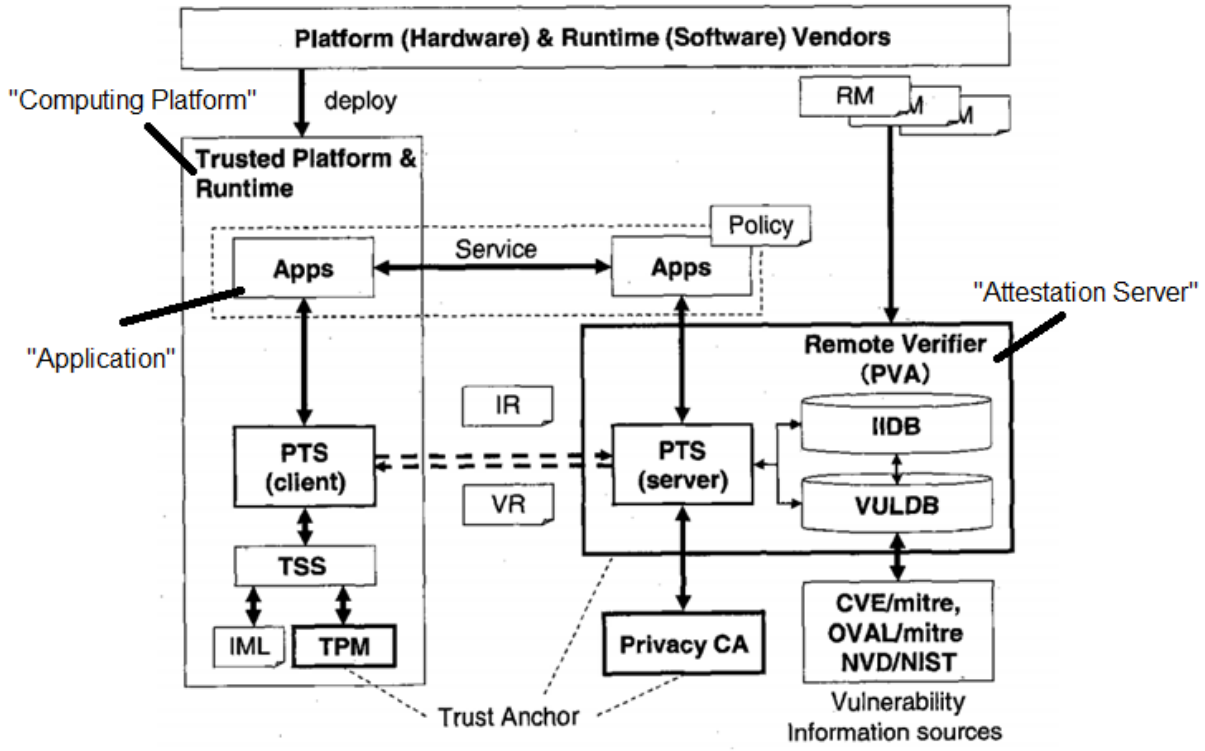


Fig. 1 PTS Client/Server architecture.

(Ex. 1005 at Figure 1, annotated).

138. In short, even though Rajan refers to the component as a service provider, it is an attestation client and receives the attestation from the broker. For clarity, I will refer to "service client" to mean a client that is requesting a service such as those disclosed in Rajan and an "attestation client" to refer to a client that is requesting the attestation service disclosed in both Rajan and Munetoh. Similarly, I will simply use "service provider" (instead of server) for the provider of a service and "attestation server" to refer to a component that provides the attestation service.

139. In Rajan, service providers register with the broker and identify themselves by the functionality (services) and the attestation capabilities they provide:

> Traditional interaction pattern publish-find-bind between components in SoAs is augmented with steps for negotiating requirements monitoring

> capabilities as shown in Fig. 2. ***During the publish step, in addition to the functional properties, service providers also make the capabilities of the requirements monitor available to the broker via the trust negotiation interface. By capabilities, we mean the type of nonfunctional requirements that the monitor can verify.*** For example, a service provider that deploys a monitor similar to Barbon et al.'s [5] work would make the capabilities to verify Boolean, statistical, and timing properties available.
> (Ex. 1006 at p. 3, emphasis added).

140. When a service client requests a service, the broker identifies services that both provide the necessary functionality as well as the necessary attestation. As Rajan explains, "[t]he broker responds to [the service client's] request with a list of only those services that satisfy both functional requirements as well as provide monitoring capabilities for nonfunctional requirements." (Ex. 1006 at p. 4). Rajan explains that the same kind of attestation operations (e.g., recording integrity measurements in PCR registers) disclosed by Munetoh are used for proving that the service is compliant:

> During the publish step, the trust analyzer and requirements monitor send integrity measurements to the broker after mutual authentication. ***The integrity measurements are stored in the platform configuration registers (PCRs) of the TPM using the SHA1 hashing algorithm***. The key idea is to store integrity measurements as successive hash values. For example, let us assume that integrity measurements of the trust analyzer resulted in a hash value of $hash_{TA}$ in the PCR. The integrity measurement for the requirements monitor now uses $hash_{TA}$ as its base value.
> (Ex. 1006 at p. 4, italicized $hash_{TA}$ original, other emphases added).

141. As also explained previously, Rajan discloses how the attestation capabilities are used for monitoring specific programs (or applications) while they are running with fine-grained, operation-specific validation.

> In this work, we focus on the trace-based requirements monitor… In a trace-based monitor, requirements are specified as logical formulas ***over program traces***. An example of such a trace-based requirement for the payment processing service could be, "***in no trace of the service implementation, the argument credit card number directly or indirectly flows to a persistence-related method, e.g., File.write(...)***."
> (Ex. 1006 at pp. 5-6, emphasis added).

142. In summary, Rajan discloses providing attestation as to what kinds of operations an application is performing at runtime.

143. I have already explained that a POSITA would have been motivated to combine Munetoh and Rajan in order to obtain these advantages in Section IX.A. I incorporate that analysis into this section by reference.

144. Thus, I conclude that Munetoh and Rajan each discloses limitation 1[a].

2. Limitation 1[b]

145. Limitation 1[b] of the '441 patent reads:

receiving, by the attestation server remote from the computing platform:

146. Munetoh discloses limitation 1[b]. Rajan also discloses limitation 1[b].

147. Munetoh discloses receiving by an attestation server attestation data from remote computing platforms. As explained in Section VIII.A., Munetoh discloses a Platform Verification Authority (PVA). The PVA contains the server-side PTS: "[t]he server-side PTS is an entity which verifies the IRs and returns a Verification Result (VR)." (Ex. 1005 at p. 5). See also Munetoh Figure 3, reproduced hereinbelow.

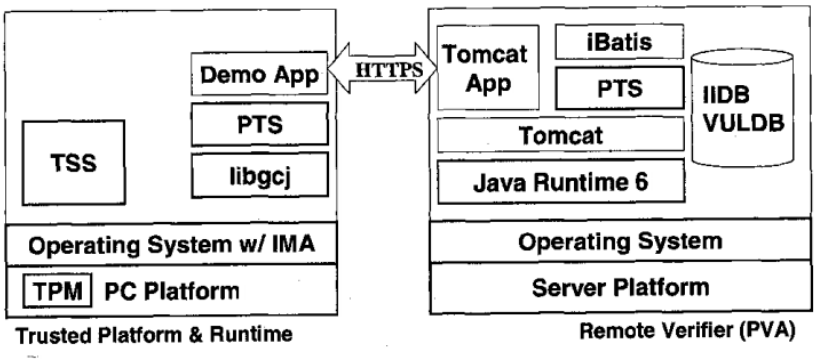


Fig. 3 Our prototype implementation of PTS and demo app.

(Ex. 1005 at Figure 3).

148. Munetoh explains that the attestation client transmits "Integrity Reports" (IR) to the server. "In our prototype, the IR and the verification result (both in XML) are base-64 encoded and exchanged in the HTTPS message body." (Ex. 1005 p. at 6).

149. Munetoh further explains that the integrity data, called "Integrity Reports" (or IRs), is sent by a trusted platform via the "client-side PTS."

> ***The client-side PTS is an entity which generates Integrity Report (IR), by collecting information from the Integrity Measurement Log (IML) and the PCR values signed by the TPM with an AIK, through TCG Software Stack (TSS)***."
> (Ex. 1005 at p. 4, emphasis added).

150. The client-side PTS can be a trusted application inside the TCB or outside the TCB of the trusted platform. Either way, the integrity data is collected.

> Figure 2 shows two different approaches for the implementation of the client-side PTS. Figure 2(a) shows the PTS implementation in our prototype on ordinary Linux. In (a), the PTS runs as an application in the user space, and thus is not part of the trusted computing base (TCB). In this case, the TCB has an integrity measurement function and needs to cooperate with the PTS, and userland applications, include the PTS, are measured by the OS Kernel."
> (Ex. 1005 at p. 5)

151. See also Munetoh Figure 2, reproduced hereinbelow:

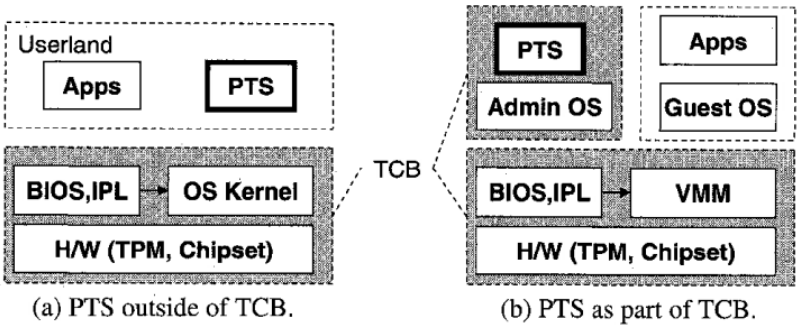


Fig. 2 Two type of PTS implementation.

(Ex. 1005 at Figure 2).

152. Munetoh's Figure 1, reproduced hereinbelow, also illustrates the IR data transferred from the client-side PTS to the server-side PTS.

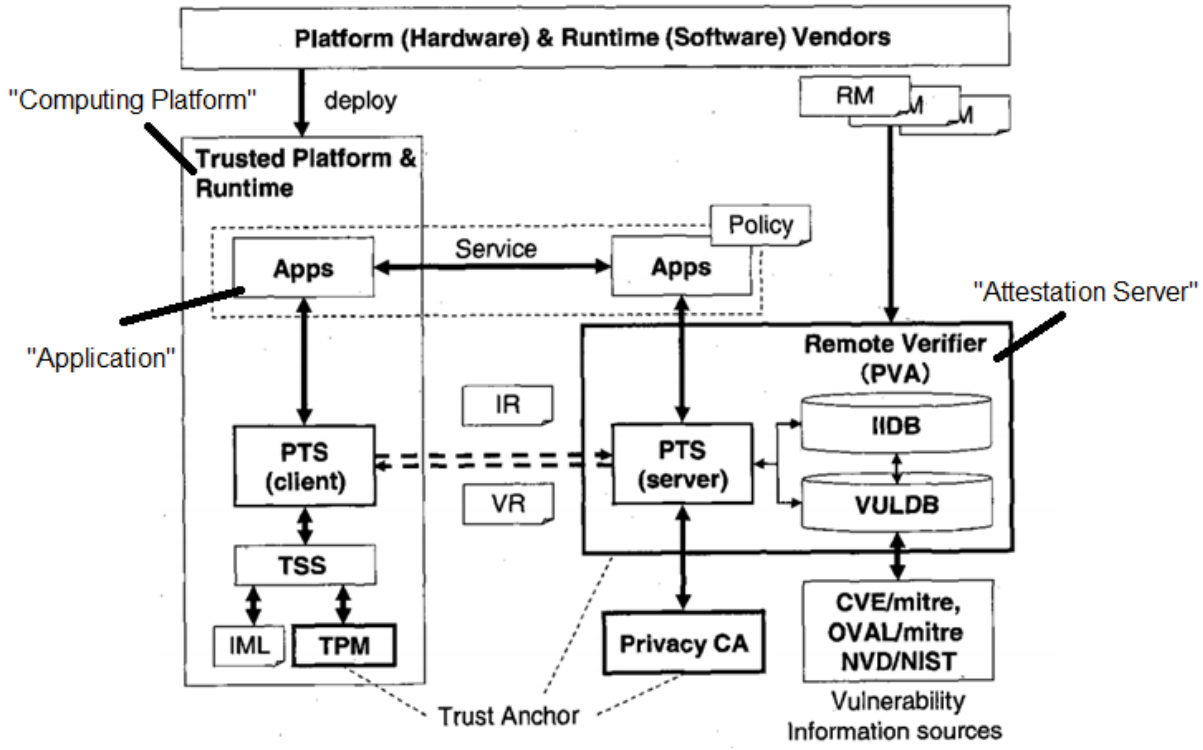


Fig. 1 PTS Client/Server architecture.

(Ex. 1005 at Figure 1, annotated).

153. Rajan also discloses receiving attestation data by an attestation server. This is clearly shown in Figure 2, of which an annotated version is reproduced hereinbelow, of Rajan that shows the "Provider" of the service (e.g., provided by a server) as sending the integrity measurements to the broker (which provides the attestation service).

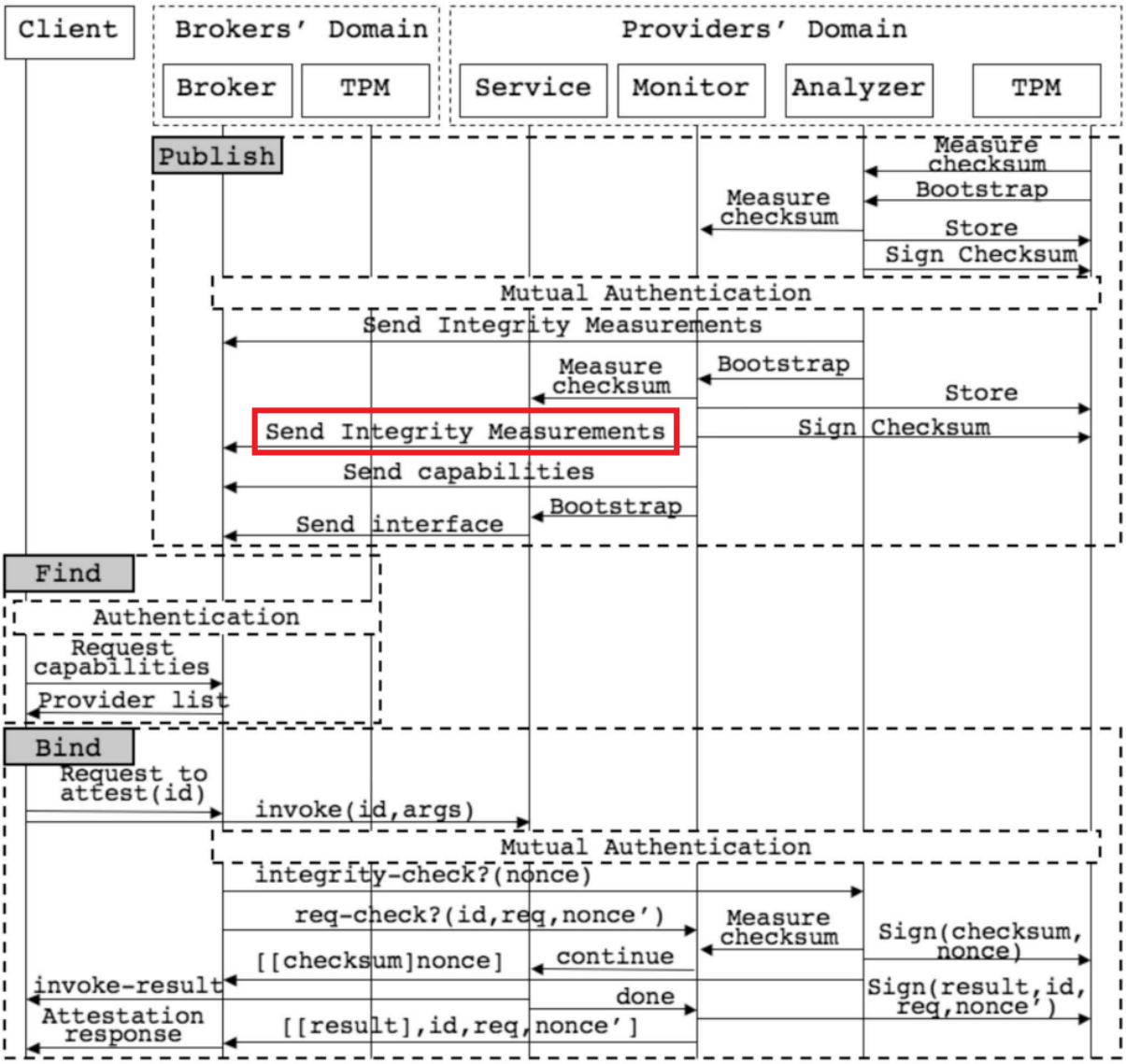


Fig. 2. Overview of Interaction between Components.

(Ex. 1006 at Figure 2, annotated).

154. Rajan explains, as illustrated in Figure 2 above, that service providers register with the broker and identify themselves by the functionality (services) and the attestation capabilities they provide:

> Traditional interaction pattern publish-find-bind between components in SoAs is augmented with steps for negotiating requirements monitoring capabilities as shown in Fig. 2. ***During the publish step, in addition to the functional properties, service providers also make the capabilities of the requirements monitor available to the broker via the trust negotiation***

> ***interface. By capabilities, we mean the type of nonfunctional requirements that the monitor can verify.*** For example, a service provider that deploys a monitor similar to Barbon et al.'s [5] work would make the capabilities to verify Boolean, statistical, and timing properties available. (Ex. 1006 at p. 3, emphasis added).

155. Then, once the integrity measurements are taken, Rajan discloses transmitting them over a secure channel:

> The cryptographic capabilities of the TPMs are leveraged to establish a secure communication channel between the service provider and the broker. ***This channel is used to convey the integrity measurements to the broker***. (Ex. 1006 at p. 5, emphasis added).

156. I have already explained that a POSITA would be motivated to combine Munetoh and Rajan with respect to limitation 1[a]. A POSITA would have recognized that the disclosures of Munetoh and Rajan, with respect to limitation 1[b], are compatible. A POSITA could rely on either reference to satisfy this limitation.

157. Thus, Munetoh discloses limitation 1[b]. Rajan also discloses limitation 1[b].

3. Limitation 1[c]

158. Limitation 1[c] of the '441 patent reads:

> *a runtime execution context indicating attributes of the application at runtime, wherein the attributes comprise one or more executable file binaries of the application and loaded components of the application; and*

159. Munetoh discloses limitation 1[c]. Rajan also discloses limitation 1[c].

160. Munetoh discloses a runtime execution context that includes one or more executable file binaries of the application and loaded components of the application. As I have discussed with respect to previous limitations, Munetoh discloses that the client-side PTS of a trusted platform generates an Integrity Report (IR) that includes the PCR values signed by the TPM.

> ***The client-side PTS is an entity which generates Integrity Report (IR), by collecting information from…the PCR values signed by the TPM with an AIK, through TCG Software Stack (TSS)***."
> (Ex. 1005 at p. 4, emphasis added).

161. Munetoh explains that the PCR values in the IR contains information about the executable binaries that are running (i.e., "runtime"). For example, Munetoh explains that the IR contains platform integrity measurements that track the software starting from the BIOS boot process:

> Platform integrity measurement consists of multiple phases of measuring and storing the fingerprints of the hardware and software components that make up a platform. Integrity measurement can be categorized into two types: ***1) the TCG trusted bootstrap and 2) other integrity measurements built on top of the trusted bootstrap***.
> (Ex. 1005 at p. 2, emphasis added).
>
> TCG defines the trusted bootstrap process [13], ***that comprises an iterative process of "measurement", loading, and execution of software components. When the system is powered-on, the immutable initial bootstrap code (such as the BIOS boot block) measures the next component and stores the measurement in the TPM before transferring control to the next component. In subsequent steps, each software component recursively measures the next component and records the measurements in the TPM, until the operating system is loaded***.
> (Ex. 1005 at p. 2, emphasis added).

162. As explained above, the results of these measurements are stored in Platform Configuration Registers (PCRs). The PCRs start with getting the measurements for the operating system.

> Each measurement uses a SHA-1 hash value of the binary image of a component, and stores it into a Platform Configuration Register (PCR). (Ex. 1005 at p. 2)
>
> After the OS is loaded and initialized in a trusted bootstrap process, the PCRs will contain some predictable values, provided that the platform runs a known set of components and they are intact.
> (Ex. 1005 at p. 3).

163. After the trusted bootstrap is completed, additional data (such as applications) are also measured. Munetoh first explains the concept:

> Integrity measurement can be categorized into two types: 1) the TCG trusted bootstrap and 2) ***other integrity measurements built on top of the trusted bootstrap***.
> (Ex. 1005 at p. 2, emphasis added).

> The Integrity Measurement Architecture [14] supports such measurements for the Linux kernel, and enhances the role of the TPM for measuring not only the static state of a system but also the dynamic state. Several proposals have been made [9], [10], [12] to measure additional states, including the configuration of the OS, middleware, and applications, to further enhance the role of TPM to the upper layers of the software stack. ***Care needs to be taken, though, that a transitive trust chain is maintained from the CRTM all the way to the component being measured.***
> (Ex. 1005 at p. 3, emphasis added).

164. Munetoh disclosed proving the applications (e.g., "user land executables") in the IR as discussed above.

> We used a commercially available PC with a TPM chip and KNOPPIX 5.1.1 for Trusted Computing Geeks [27] as the target platform… This modified KNOPPIX supports the trusted boot, including the transitive trust chain which covers the ***CRTM, BIOS, bootloader, operating system, and the user land executables.*** The Integrity of the user land executables and libraries are measured with the Linux-IMA. ***Thus, the IR generated by the client-side PTS contains all of the components that are running or ran on the target platform.***
> (Ex. 1005 at p. 8).

165. The IRs, containing the signed PCR values, are sent from the client-side PTS to the server-side PTS.

> Platform Integrity Reporting, or remote attestation, is a mechanism defined in the TCG specification [13] t***o report integrity measurements stored in the PCRs in a reliable manner***. In the attestation process, the TPM signs the PCR values and an external 160-bit challenge (such as a nonce from a challenger to avoid replay attacks) using an RSA private key, the confidentiality of which is protected by the TPM. The attestation is an atomic, protected operation and the attestation signature cannot be forged by malicious software. Therefore, if the TPM is properly designed and implemented to adhere to the TPM specifications, and the platform,

> including the initial bootstrap code, is properly integrated with the TPM, ***a remote verifier can have confidence in the integrity measurements reported by the TPM***.
> (Ex. 1005 at p. 3, emphasis added.).
>
> ***The client-side PTS is an entity which generates Integrity Report (IR), by collecting information from the Integrity Measurement Log (IML) and the PCR values signed by the TPM with an AIK, through TCG Software Stack (TSS)***."
> (Ex. 1005 at p. 4, emphasis added).

166. Rajan also discloses limitation 1[c]. Rajan discloses transmitting an execution context that includes information about the software stack of the service including the monitor.

> This trace, along with static checksum of the software stack, is sent to the trusted third party [broker] (step 7). To remind the reader, the static checksum is created by computing the SHA1 hash of the software stack up to the requirements monitor and is essential to validate the current state of the service implementation (step 5). This hash is signed using the AIK on the TPM of the service provider (step 6).
> (Ex. 1006 at p. 7).

167. Rajan's checksum is a runtime execution context that includes attributes about that requirements monitor, which is a loaded component of the application. That is, the requirements monitor is one component of the software stack for which the checksum is computed.

168. I have already explained that a POSITA would be motivated to combine Munetoh and Rajan with respect to limitation 1[a]. A POSITA would have recognized that the disclosures of Munetoh and Rajan, with respect to limitation 1[c], are compatible. In the combined teachings, a POSITA could rely on either reference to satisfy this limitation.

169. Thus, Munetoh discloses limitation 1[c]. Rajan also discloses limitation 1[c].

4. Limitation 1[d]

170. Limitation 1[d] of the '441 patent reads:

a security context providing security information about the application, wherein the security information comprises an execution analysis of the one or more executable file binaries and the loaded components;

171. Munetoh in view of Rajan discloses limitation 1[d].

172. Munetoh discloses transmitting a security context that provides security information about the application including information about the execution of the one or more executable file binaries and the loaded components. Specifically, Munetoh explains that, in addition to the PCR values that measure the components that are loaded, additional data may be sent in the "integrity measurement log," including a list of components measured by the TPM and how they are put together ("composed").

> The platform may also send additional information in an attestation, such as the integrity measurement log, ***which includes a list of components loaded on the platform and measured by the TPM, and how these components are composed***.
> (Ex. 1005 at p. 3, emphasis added).

> The client-side PTS is an entity which generates Integrity Report (IR), by collecting information from the ***Integrity Measurement Log (IML)*** and the PCR values signed by the TPM with an AIK, through TCG Software Stack (TSS).
> (Ex. 1005 at p. 4, emphasis added).

173. Munetoh discloses a Finite State Machine (FSM) that processes the transmitted IML in order to determine the current state of the software running on the system. As Munetoh explains,

> … we have extended the format of the RM to incorporate the behavior model, allowing the PTS to use it when verifying the IRs. ***Intuitively, the behavior model is a finite state machine (FSM) that can consume an IML***. The PCR values and the IML in the IR are generated as a result of running the trusted platform, while the behavior model describes how they are measured, and how they should be verified.

> The FSM is a representation of the trusted-bootstrap and subsequent measurement processes of the platform. ***Each state in the behavior model represents a platform state, and each transition represents the integrity of the component that is loaded and executed***.
> (Ex. 1005 at p. 6, emphasis added)

174. As explained with respect to limitation 1[b], this information is included in the IR and is sent from the client-side PTS of the trusted platform to the server -side PTS of the PVA.

175. Moreover, Munetoh also discloses that the running components themselves may also transmit assertions about their operations.

> Integrity Report Schema represents a snapshot of the platform integrity measurement, including the IML, and signed by the attestation signature of the TPM to attest to the authenticity of the report. ***The IR may also include assertions made by the components running on the client platform.***
> (Ex. 1005 at p. 5, emphasis added).

176. I have already explained in Section IX.A. that a POSITA would have been motivated to combine Rajan with Munetoh in order to obtain the more fine-grained assertions about the execution of applications using a trace-based mechanism. A POSITA would have recognized that Rajan's trace-based execution analysis, as discussed in Section VIII.A., are "assertions made by the components running on the client platform" as disclosed by Munetoh. Rajan discloses transmitting the trace-based assertions, which is a runtime execution analysis of the service application.

> In this work, we focus on the trace-based requirements monitor… In a trace-based monitor, requirements are specified as logical formulas ***over program traces***. An example of such a trace-based requirement for the payment processing service could be, "***in no trace of the service implementation, the argument credit card number directly or indirectly flows to a persistence-related method, e.g., File.write(...)***."
> (Ex. 1006 at pp. 5-6, emphasis added).

177. Then, once the integrity measurements are taken, Rajan discloses transmitting them over a secure channel:

> The cryptographic capabilities of the TPMs are leveraged to establish a secure communication channel between the service provider and the broker. ***This channel is used to convey the integrity measurements to the broker***. (Ex. 1006 at p. 5, emphasis added).

178. As discussed with respect to limitation 1[c], Rajan discloses sending the trace data (i.e., the execution analysis) along with the static attestation of the software stack (i.e., the runtime execution context).

> Each transaction is validated by executing a series of steps shown in Procedure 2. First, the trust analyzer receives a nonce from the broker. Standard mechanisms for avoiding replay attacks and verifying the authenticity of the trusted third party are used here. A key part in this authentication session is played by the service provider's TPM, which signs the message containing the nonce with its key. ***During the computation of the results by the service implementation, a trace is generated and maintained by the requirements monitor (steps 2-4).***
>
> ***This trace, along with static checksum of the software stack, is sent to the trusted third party [broker] (step 7).*** To remind the reader, the static checksum is created by computing the SHA1 hash of the software stack up to the requirements monitor and is essential to validate the current state of the service implementation (step 5). This hash is signed using the AIK on the TPM of the service provider (step 6).
> (Ex. 1006 at p. 7).

179. See also, e.g., Rajan's Procedure 2, reproduced hereinbelow:

> **Procedure 2.** Bind step for providers' end.
> **Input:** Broker Address: brokerAddr, Service Id: Id
> 1. nonce = recv(brokerAddr, Id);
> 2. startTraceCollection();
> 3. continueService();
> 4. trace = collectTrace();
> 5. checksum = computeSHA1Hash();
> 6. response = $sign_{AIK}(trace, nonce, checksum)$;
> 7. send(brokerAddr, Id, response);

(Ex. 1006 at p. 6).

180. I have already explained that a POSITA would be motivated to combine Munetoh and Rajan with respect to limitation 1[a]. A POSITA would have been motivated to combine

Rajan's execution analysis with Munetoh's IML security context. As discussed in Section IX.A., a POSITA would have been motivated to take advantage of Rajan's fine-grained, operation-level attestation. I incorporate my analysis into this section by reference.

181. Thus, Munetoh in view of Rajan discloses limitation 1[d].

5. Limitation 1[e]

182. Limitation 1[e] of the '441 patent reads:

> *generating, by the attestation server, a report indicating security risks associated with the application based on the received runtime execution context and the received security context, as an attestation result; and*

183. Munetoh in view of Rajan discloses limitation 1[e].

184. Munetoh discloses generating, by the attestation server, a report indicating security risks associated with the application based on the received runtime execution context and the received security context. Specifically, Munetoh discloses generating a report of the integrity measurement and the validation thereof through a Platform Validation Authority, or PVA.

> In this paper, we introduce the concept of the Platform Validation Authority (PVA), ***a trusted third party which verifies the composition of each target platform reported in remote attestations through the use of the PTS.*** The PVA complements the role of the current CA in the PKI by adding a new feature of platform attestation.
> (Ex. 1005 at p. 2, emphasis added).

185. Munetoh explains that the PVA also contains Reference Manifests (RM) and a vulnerability database for analyzing the security of the attested platform.

> The server-side PTS is an entity which verifies the IRs and returns a Verification Result (VR). The result is one of three values: valid, invalid, or unverified, which means that the IR is verified as trusted, the IR is verified to be invalid, or there is not enough information for making a decision, respectively. ***The server-side PTS verifies IRs against the RM. The integrity of the RM, which represents the expected platform integrity***

> ***metrics, is protected by the signature by the issuer, e.g., the platform vendor or a PVA.***
>
> The vendors release software with RMs and the software is deployed into the target platforms. ***The PVA maintains the integrity information database (TIDB) and the vulnerability database (VULDB) for improving performance of validation.*** The IIDB keeps digests and package information obtained from RMs. The latest vulnerability information are retrieved from external sources, such as CVE[16], OVAL [17] and NVD[18].
> (Ex. 1005 p. 5, emphasis added).

186. Munetoh explains that the PVA uses the RMs to compute integrity information. Next, the PVA uses a Finite State Machine (FSM) to process an Integrity Measurement Log (IML) to verify the current state.

> … we have extended the format of the RM to incorporate the behavior model, allowing the PTS to use it when verifying the IRs. ***Intuitively, the behavior model is a finite state machine (FSM) that can consume an IML***. The PCR values and the IML in the IR are generated as a result of running the trusted platform, while the behavior model describes how they are measured, and how they should be verified.
>
> The FSM is a representation of the trusted-bootstrap and subsequent measurement processes of the platform. ***Each state in the behavior model represents a platform state, and each transition represents the integrity of the component that is loaded and executed***.
> (Ex. 1005 at p. 6, emphasis added).

187. Once the state of the remote system is known, the PVA uses an "integrity database", including a vulnerabilities database, to determine which, if any, known vulnerabilities are present on the remote system.

> Our server-side PTS implementation contains the integrity database that integrates the expected integrity metrics of software packages ***with vulnerability information obtained from external security advisories***… Information security vulnerabilities ***can be identified by CVE (Common Vulnerabilities and Exposures) Identifiers [16] managed and provided by MITRE Corporation…*** The PVA links the vulnerability information and the integrity information collected from Reference Manifests and verifies the security state from the latest components.
> (Ex. 1005 at p. 7, emphasis added).

188. In more detail, Munetoh discloses using "vulnerability scores" from publicly available sources: "The NVD (National Vulnerability Database) [18] provides CVSS scores for most of the CVE entries, and thus each security advisory can be associated with a vulnerability score." (Ex. 1005 at p. 7). These scores are used when evaluating software for vulnerabilities:

> We inspect platforms with reference to these vulnerability information sources. Although there are vendor specific vulnerability checking tools, we use OVAL (Open Vulnerability and Assessment Language) [17], which is a generic open framework for representing the vulnerability test processes. OVAL can describe the test cases to detect the presence of vulnerabilities. From the package management perspective, OVAL describes the versions of the base software distributions and how to check the vulnerable component versions. In addition, OVAL can describe various conditions to analyze the vulnerabilities, that can be evaluated by the OVAL interpreter. ***An OVAL interpreter, which is customized for each operating system, collects the detailed states of the platform, tests them against the OVAL definitions, and reports the results of the tests.***
> (Ex. 1005 at p. 7, emphasis added).

189. Furthermore, Munetoh is explicit that these results identify vulnerabilities in particular applications (e.g., iceweasel-2.0.0.1):

> When a user load [sic] the KNOPPIX and tries to validate it by using the PVA, The IML is verified against the RMs and the latest states are also verified according to the data in these databases. ***For example, if the hash value "e8df910d0bd25ebe7a0b..." is in the IML, we will know that the hash value is for the file "/usr/lib/iceweasel/firefox-bin" and a part of the "iceweasel-2.0.0.1+dfsg-l" package.*** This package has several vulnerabilities according to DSA-1308, such as CVE-2007-2871, 2870, 2869, 2868, 2867, and 1362. The CVSS has a range of scores from 4.3 to 9.3 in the relevant CVE entries.
> (Ex. 1005 at p. 8, emphasis added).

190. The final report uses the vulnerability analysis to identify the security risks associated with the state of the system.

> ***The report, which describes the presence of vulnerabilities and the degrees of security risks, is generated by verifying the integrity of the files and evaluating them against the vulnerability information.***
> (Ex. 1005 at p. 7, emphasis added).

191. Munetoh also discloses that the verification result format includes a verification status as well as explanations for the status.

> **Validation Result Schema:** communicates a verification result from the verifier (e.g., the server-side PTS) to the client. The result is one of the following: VALID, INVALID, or UNVERIFIED. Also it contains ReasonStrings for a detailed reason when the result is not VALID. To validate an IR to derive a result, the PVA and the relying party need to share a policy that describes the detailed conditions of the target platform. The issue will be discussed in Sect. 4.3.
> (Ex. 1005 at p. 5).

192. Thus, the Validation Result comprises a report indicating static security risks associated with the application.

193. Alternatively, the final vulnerability report itself (which, as cited hereinabove, "describes the presence of vulnerabilities and the degrees of security risks" and is part of the Validation Result) comprises a report indicating static security risks associated with the application. As explained above, such final report is analyzing "security risks" associated with the files that comprise the application and thus contains the information required by the claimed "attestation result."

194. As discussed with respect to limitation 1[d], I explained that it would have been obvious for a POSITA to combine Rajan's trace-based execution analysis with Munetoh's IML-based security context. Rajan discloses that the broker determines the attestation result for the service based on the trace-based security information transmitted from the service to the broker as well as the static attestation of the software stack. Rajan explains that the attestation result of the Broker is calculated during runtime based on both the static hash of the data and the trace data of the application:

> During the service, the trust analyzer [of the service] monitors and creates a trace of the execution containing the field and method information

> identified during the initialization phase. The authentication server is responsible for verifying this trace. To that end, ***after receiving the signed data from the provider***… if the data from the provider and its signature do not match, it represents corruption of the provider's information and, hence, a violation of trust… ***The client is notified of such violations (steps 6, 10)***.
>
> ***Upon successfully verifying the data and the nonce, the next step is to verify the trace (step 12)***. The traces from two different executions may differ due to modifications of service implementation as well as differences in input values. Thus, exact equivalence may not be used to report violations. Instead, we use the trace received from the provider to check if it contains a sequence that violates the requirements. In our example, persistence and serialization of data are assumed to be a violation of requirements; ***thus, the presence of an API call that persists or serializes data will be reported as violation…***
>
> Finally, if all checks pass, the system configuration is checked. If any of these checks fail, the clients are notified of a violation; ***otherwise, an assurance is sent to the client***.
> (Ex. 1006 at p. 7).

195. As also illustrated in this citation, the result that is sent to the client is a notification of violations of the agreed upon requirements. This indicates a security risk associated with the application.

196. A POSITA would have been motivated to combine Rajan's analysis of dynamic vulnerabilities observed while the application is running with Munetoh's analysis of static vulnerabilities determined from analyzing files on disk. I have already explained that a POSITA would have been motivated to obtain fine-grained, operation-level attestation information. It would be equally obvious to include the result of analyzing this information in the final validation result.

197. Thus, I conclude that Munetoh in view of Rajan discloses limitation 1[e].

6. Limitation 1[f]

198. Limitation 1[f] of the '441 patent reads:

sending, by the attestation server, the attestation result associated with the application.

199. Munetoh and Rajan each discloses limitation 1[f].

200. Munetoh discloses sending, by the attestation server, the attestation result associated with the application. As discussed in limitation 1[e], Munetoh discloses generating, by the attestation server, a report indicating the security risks of the remote application as the attestation result and it would have been obvious for a POSITA to include the runtime risks determined by the trace-based attestation disclosed by Rajan. Munetoh discloses that the results are recorded in the Verification Result (VR), which included the status of the verification and the reasons for the status.

201. Munetoh also discloses transmitting the VR back to the trusted platform that transmitted the Integrity Report. "In our prototype, the IR and the verification result (both in XML) are base-64 encoded and exchanged in the HTTPS message body." (Ex. 1005 at p. 6). See also, Munetoh Figure 3, reproduced hereinbelow.

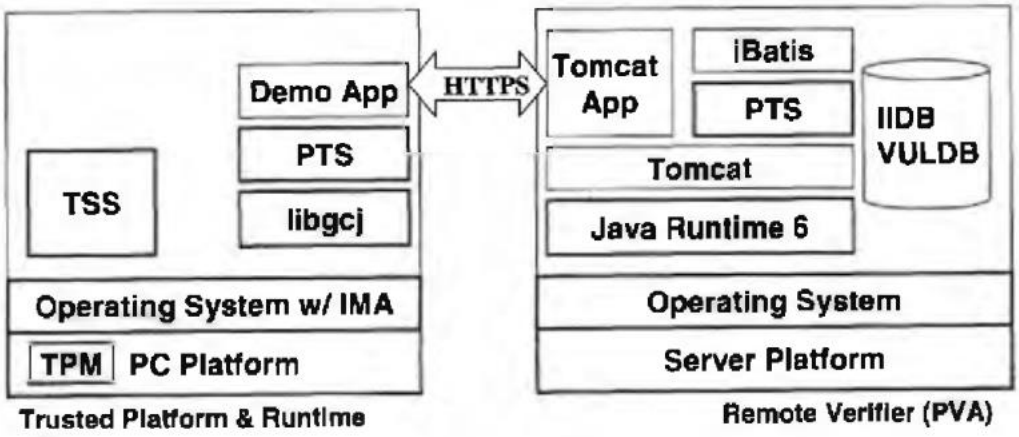


Fig. 3 Our prototype implementation of PTS and demo app.

(Ex. 1005 at Figure 3).

202. See also, Figure 1 of Munetoh, reproduced hereinbelow.

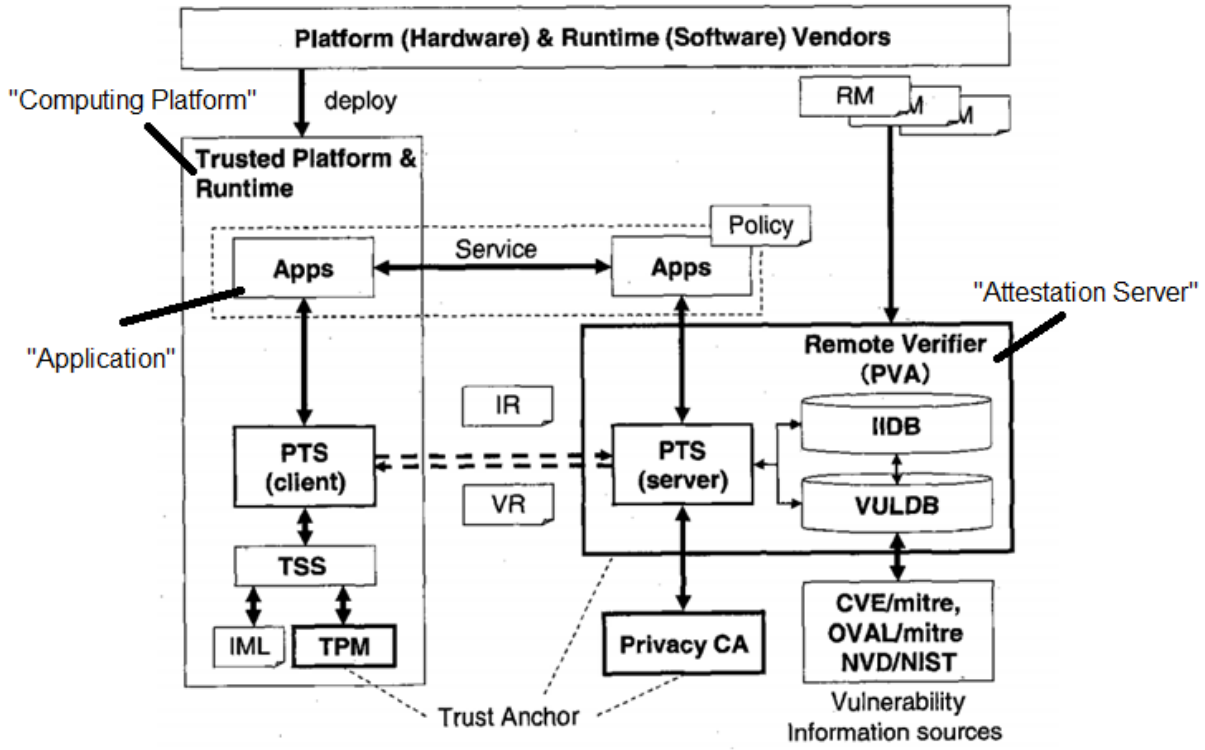


Fig. 1 PTS Client/Server architecture.

(Ex. 1005 at Figure 1, annotated).

203. Munetoh also describes the "Validation Result Schema" as explicitly having the purpose of communication of results:

> **Validation Result Schema:** communicates a verification result from the verifier (e.g., the server-side PTS) to the client.
> (Ex. 1005 at p. 5).

204. As I explained with respect to limitation 1[e], the claimed "attestation result" may alternatively be the vulnerability report itself. Such report may be sent as part of the "Validation Result". (*See* Ex. 1005 at p. 5: "Also it contains ReasonStrings for a detailed reason when the result is not VALID."). Alternatively, a POSITA would have understood such report to have been transmitted to the client, as in Munetoh's disclosure: "An OVAL interpreter, which is customized

for each operating system, collects the detailed states of the platform, tests them against the OVAL definitions, and ***reports the results of the tests.***" (Ex. 1005 at p. 7).

205. Rajan further discloses sending the validation result to different parties other than the computing platform sending the attestation data. Rajan's disclosures include transmitting the data to clients of the service:

> During the service, the trust analyzer [of the service] monitors and creates a trace of the execution containing the field and method information identified during the initialization phase. The authentication server is responsible for verifying this trace. To that end, after receiving the signed data from the provider… if the data from the provider and its signature do not match, it represents corruption of the provider's information and, hence, a violation of trust… ***The client [of the service] is notified of such violations (steps 6, 10)***.
>
> Upon successfully verifying the data and the nonce, the next step is to verify the trace (step 12). The traces from two different executions may differ due to modifications of service implementation as well as differences in input values. Thus, exact equivalence may not be used to report violations. Instead, we use the trace received from the provider to check if it contains a sequence that violates the requirements. In our example, persistence and serialization of data are assumed to be a violation of requirements; ***thus, the presence of an API call that persists or serializes data will be reported as violation…***
>
> Finally, if all checks pass, the system configuration is checked. If any of these checks fail, the clients are notified of a violation; ***otherwise, an assurance is sent to the client [of the service]***.
> (Ex. 1006 at p. 7, emphasis added).

206. I have already explained in Section IX.A. that a POSITA would have been motivated to combine Rajan with Munetoh in order to take advantage of this more flexible system architecture. I incorporate my analysis into this section by reference.

207. Thus, I conclude that Munetoh and Rajan each discloses limitation 1[f].

208. Thus, I conclude that Munetoh in view of Rajan renders Claim 1 obvious to a POSITA.

C. Claim 2 is rendered obvious by Munetoh in view of Rajan

209. Claim 2, which depends from Claim 1, is rendered obvious to a POSITA by Munetoh in view of Rajan.

1. Limitation 2[a]

210. Limitation 2[a] reads:

The method of claim 1, further comprising:

generating, by the attestation server, an application artifact as a reference for changes in a subsequent execution context; and

211. Munetoh in view of Rajan discloses limitation 2[a]. As discussed above, Munetoh in view of Rajan discloses all limitations of Claim 1, from which Claim 2 depends.

212. Rajan discloses generating at the broker, which is the attestation server, a "reference" hash of the software stack (i.e., up to the requirements monitor). This reference hash is an application artifact. Rajan explains that this hash is generated by the broker during the Publish step. For example, Rajan discloses "Procedure 3" for the Broker that, in step 8, computes the SHA1 hash of the software stack.

Procedure 3. Publish step for brokers' end.
Input: Provider Address: providerAddr, Service Id: Id
Input: Property as Criterion: sliceCriterion

```
1. impl = recvImpl(providerAddr, Id);
2. reqMonitor = recvMonitor(providerAddr, Id);
3. confFiles = recvConf(providerAddr, Id);
4. progPoints = slice(impl, sliceCriterion);
5. instImpl = instrument(impl, progPoints);
6. progTrace = genTrace(instImpl, reqMonitor, confFiles);
7. store(progTrace);
8. checksum = computeSHA1Hash();
9. store(checksum);
10. send(providerAddr, Id, progPoints);
```

(Ex. 1006 at p. 7).

213. Rajan explains further, with respect to this procedure, "[d]uring this [publish] step, reference measurements are also stored for the software stack upto [sic] the requirements monitor (steps 8-9)." (Ex. 1006 at p. 7).

214. This reference hash is used during service execution to ensure that the software stack of the service provider has not changed (i.e., remains "immutable"):

> During service execution, the broker and the provider must communicate to verify that: ***the provider and the broker are indeed the same principals that agreed to the contract during the publish step, the requirements monitor as well as the software stack on the provider's end is not compromised***, and the trace provided by this monitor does not violate the properties desired by the client.
>
> The trust analyzer component, which derives its root of trust from the TPM on the provider's side, serves to verify the integrity of the requirements monitor. ***It provides a report that includes an immutable integrity measurement of the local architecture along with a time stamp to indicate the freshness of the measurement***.
> (Ex. 1006 at p. 6, emphasis added)

215. To the extent that Rajan does not explicitly disclose comparing the reference hash against subsequent hashes, it would have been obvious to do so. A POSITA would have understood "reference measurements" to be used for comparison against the subsequent measurements sent during service execution. For example, an article in IEEE Pervasive Computing from 2007 describes using hash-based "reference measurements" in exactly this way and using the Integrity Measurement Architecture mentioned by Munetoh:

> Trust-Sniffer builds on the implementation of Integrity Measurement Architecture for Linux. IMA checks the system software stack's integrity by computing a SHA-1 hash over an executable's contents when it's loaded. This SHA-1 hash is called a *measurement*…
>
> Trust-Sniffer validates an application by ***comparing its sample measurement, obtained when the application is loaded, to a known list of reference measurements generated from trusted applications. A mismatch is an inequality between a sample measurement and every reference measurement in the list.*** In contrast to attestation, where measurements

> detect untrusted software after it has been loaded, validation refers to detection of untrusted software before it's used.
> (Ex. 1011 at p. 3, italics original, additional emphasis added).

216. I have already explained in Section IX.A. and with respect to limitation 1[a] that it would have been obvious to a POSITA to combine Munetoh and Rajan. I incorporate my analysis into this section by reference.

217. Thus, I conclude that Munetoh in view of Rajan discloses limitation 2[a].

2. Limitation 2[b]

218. Limitation 2[b] reads:

> *sending the generated application artifact such that subsequent changes to the runtime execution context are tracked based on the generated application artifact.*

219. Munetoh in view of Rajan discloses limitation 2[b].

220. As discussed with respect to limitation 2[a], Rajan discloses generating a reference hash for the software stack up to the requirements monitor. I also explained that, based on this hash, subsequent changes to the runtime execution context of the software stack, such as unauthorized changes, are tracked based on the reference hash, which is the generated application artifact.

221. As discussed with respect to limitation 1[e], Munetoh discloses sending Validation Results (VR) to the party seeking attestation. A POSITA would have been motivated to combine this with Rajan's disclosures of sending the current static hash. (*See* my analysis of limitation 1[e] and Ex. 1006 at p. 7). Munetoh discloses that ongoing authentication is expected between client and server as indicated by having the client do its own checks to reduce networking load for obvious errors (i.e., that would cause attestation to fail).

> We use the same PTS component for the client and the server as shown in Fig. 3. Thus the client-side PTS may also verify the IR against the RM. The verification by the client-side PTS filters out obviously erroneous IRs without needing to send them to the PTS server. Since each IR can become very large (e.g., 100 KB or more), the client-side verification reduces the unnecessary traffic to the server.
> (Ex. 1005 at p. 6).

222. As disclosed by Munetoh, it is expected that there will be occasional failures to get a successful attestation result. It would have been obvious to include Rajan's computed hash of the attestation checksum in the VR sent back to the attestation client so that the client could determine what the current state is at the attestation server and potentially fix errors if there are any. This is also supported by Rajan which, in Procedure 3, sends data to the attestation client (i.e., the Provider in Rajan's case) immediately after computing the checksum.

> **Procedure 3.** Publish step for brokers' end.
> **Input:** Provider Address: providerAddr, Service Id: Id
> **Input:** Property as Criterion: sliceCriterion
> 1. impl = recvImpl(providerAddr, Id);
> 2. reqMonitor = recvMonitor(providerAddr, Id);
> 3. confFiles = recvConf(providerAddr, Id);
> 4. progPoints = slice(impl, sliceCriterion);
> 5. instImpl = instrument(impl, progPoints);
> 6. progTrace = genTrace(instImpl, reqMonitor, confFiles);
> 7. store(progTrace);
> 8. checksum = computeSHA1Hash();
> 9. store(checksum);
> 10. send(providerAddr, Id, progPoints);
>
> (Ex. 1006 at p. 7).

223. I understand that Patent Owner has asserted in litigation in District Court that logging data satisfies the "sending" requirement of this limitation. (Ex. 1010 at p. 20). To the extent storing data in a log file is "sending" data, Rajan also discloses "sending" the generated application artifact to storage.

224. In any case, the checksum, as discussed in limitation 2[a], is the artifact stored for subsequent comparisons. I incorporate my analysis of limitation 2[a] in this section by reference.

225. Thus, I conclude that Munetoh in view of Rajan discloses limitation 2[b].

226. Thus, I conclude that Munetoh in view of Rajan renders Claim 2 obvious to a POSITA.

D. Claim 3 is rendered obvious by Munetoh in view of Rajan

227. Claim 3 of the '441 patent is rendered obvious to a POSITA by Munetoh in view of Rajan. This claim, which depends from Claim 2, reads:

> *The method according to claim 2, further comprising digitally signing the attestation results, wherein the attributes further comprise parent-child process associations of the application.*

228. As discussed above, Munetoh in view of Rajan discloses all limitations of Claims 1 and 2, from which Claim 3 depends.

229. Munetoh discloses "comprising digitally signing the attestation results." The attestation results, as discussed with respect to limitation 1[e], is generated by the attestation server and could be either the verification result (VR) or, alternatively, the vulnerability report, "which describes the presence of vulnerabilities and the degrees of security risks". (Ex. 1005 at p. 7). Either way, Munetoh discloses digitally signing the attestation results by transmitting the results within an HTTPS message body. Munetoh explains that "In our prototype, the IR and the verification result (both in XML) are base-64 encoded and exchanged in the HTTPS message body." (Ex. 1005 at p. 5).

230. A POSITA would have known that an HTTPS message body is protected by a digital signature. As stated in one analysis from 2003:

> Https [sic] is syntactically identical to the http protocol, but the client/server interaction is performed over an encrypted Secure Sockets Layer (SSL) [now called Transport Layer Security or TLS] protocol connection. ***Along with encrypting the data, the SSL protocol creates a type of digital signature called a Message Authentication Code (MAC) using a cryptographic hash algorithm. The MAC value protects both the integrity of a message and its authenticity by allowing those who possess the secret key to detect any changes to the message content***.
> (Ex. 1009 at p. 4).

231. Munetoh also discloses "wherein the attributes further comprise parent-child process associations of the application." According to limitation 1[c], the attributes are received by the attestation server from the computing platform and are attributes of the application. As explained previously, Munetoh discloses an Integrity Measurement Log, or IML. Munetoh explains that, in addition to the PCR values that measure the components that are loaded, additional data may be sent in the "integrity measurement log," including a list of components measured by the TPM and how they are put together ("composed").

> The platform may also send additional information in an attestation, such as the integrity measurement log, ***which includes a list of components loaded on the platform and measured by the TPM, and how these components are composed***.
> (Ex. 1005 at p. 3, emphasis added).
>
> The client-side PTS is an entity which generates Integrity Report (IR), by collecting information from the ***Integrity Measurement Log (IML)*** and the PCR values signed by the TPM with an AIK, through TCG Software Stack (TSS).
> (Ex. 1005 at p. 4, emphasis added).

232. The IML is explicitly used by Munetoh for an execution analysis, which it describes as a "behavior" analysis. Munetoh discloses a Finite State Machine (FSM) that processes the transmitted IML in order to determine the current state of the software running on the system. As Munetoh explains,

> … we have extended the format of the RM to incorporate the behavior model, allowing the PTS to use it when verifying the IRs. ***Intuitively, the***

> ***behavior model is a finite state machine (FSM) that can consume an IML***. The PCR values and the IML in the IR are generated as a result of running the trusted platform, while the behavior model describes how they are measured, and how they should be verified.
>
> The FSM is a representation of the trusted-bootstrap and subsequent measurement processes of the platform. ***Each state in the behavior model represents a platform state, and each transition represents the integrity of the component that is loaded and executed***.
> (Ex. 1005 at p. 6, emphasis added).

233. A POSITA would have understood that how "components are composed" when components are running executables (as they are in this case as discussed with respect to Claim 1) would include which process launched which (i.e., parent-child process relationships). To the extent that it does not disclose this, it would have been obvious to a POSITA to include it. A POSITA would have known that process relationships are relevant for trusted computing and the relationships between processes conveys significant information with respect to determining the "health" of a platform. For example, one research paper from the 2008 time period noted:

> A process is a container for a set of resources used when executing an instance of a program. As with other operating systems, Microsoft Windows system goes through an elaborate boot up process. The boot process has a series of sequential steps since the computer is powered on. ***By understanding the details of the boot processes, we can diagnose problems that can arise during a boot.*** Our objective is to make our host system safe against any intrusions since the host starts up. The tracing abnormal behavior of system processes can achieve an execution of safe system booting and correct booting procedure. Thus, we show the execution steps of boot support files and the information on what some of the system files are for. We can represent process execution sequences of an operating system as a tree design. Figure 1 illustrates process execution sequences of Microsoft Windows NT system. The sequence of Windows boot processes is all arranged beforehand until application programs are executed. Application processes will be corresponded to a leaf node which is a node that has zero child nodes. When considering the process sequence tree, we can see that the path from a root node to a leaf node has a regular pattern.
> (Ex. 1012 at p. 3, emphasis added).

234. I have also included Figure 1 from this research paper hereinbelow:

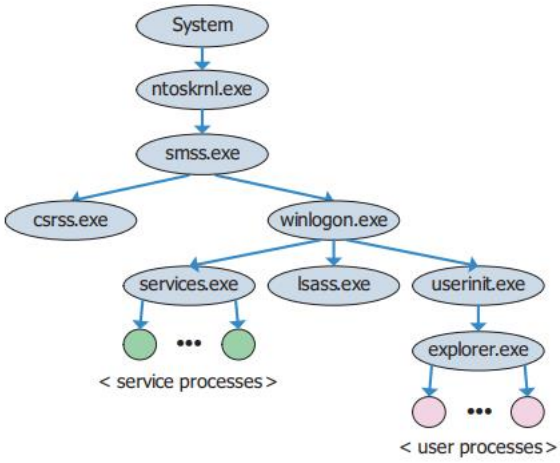


Fig. 1. The process tree of Microsoft Windows NT system

(Ex. 1012 at Figure 1).

235. In summary, a POSITA would have understood Munetoh's "how… components are composed" to include this kind of information or it would have been obvious to include it.

236. Thus, Munetoh in view of Rajan renders Claim 3 obvious to a POSITA.

E. Claim 4 is rendered obvious by Munetoh in view of Rajan

237. Claim 4 of the '441 patent is rendered obvious to a POSITA by Munetoh in view of Rajan. This claim, which depends from Claim 1, reads:

> *The method of claim 1, wherein the received security context is an introspective security context, and wherein the execution analysis is a static, dynamic, or virtual analysis of the one or more executable file binaries and the loaded components.*

238. As discussed above, Munetoh in view of Rajan discloses all limitations of Claim 1, from which Claim 4 depends. Munetoh further discloses that "the received security context is an introspective security context."

239. Munetoh discloses transmitting a "security context [that] is and introspective security context, and wherein the execution analysis is a static, dynamic, or virtual analysis of the one or more executable file binaries and the loaded components." Specifically, Munetoh explains that, in addition to the PCR values that measure the components that are loaded, additional data may be sent in the "integrity measurement log," including a list of components measured by the TPM and how they are put together ("composed").

> The platform may also send additional information in an attestation, such as the integrity measurement log, ***which includes a list of components loaded on the platform and measured by the TPM, and how these components are composed***.
> (Ex. 1005 at p. 3, emphasis added).

> The client-side PTS is an entity which generates Integrity Report (IR), by collecting information from the ***Integrity Measurement Log (IML)*** and the PCR values signed by the TPM with an AIK, through TCG Software Stack (TSS).
> (Ex. 1005 at p. 4, emphasis added).

240. A POSITA would have understood the "composition" of the components is introspective of the operation of the system because it reveals internals about how the system is constructed. That is, Munetoh must examine system internals, as reported by the TPM, in order to produce the information. Munetoh explains that the measurements require the TPM precisely because this analysis has to be securely performed internally and in a way that cannot be modified.

> The center of the TCG technology is a security module called the Trusted Platform Module (TPM), which is usually implemented as a tamper-resistant hardware module. In addition to serving as a cryptographic co-processor and protected storage for secrets and keys, the TPM is used to measure and report platform integrity in a manner that cannot be compromised even by the platform owners or the software running on it.
> (Ex. 1005 at p. 2).

241. This is introspective because it is looking at internals to perform the analysis.

242. The IML is explicitly used by Munetoh for an execution analysis, which it describes as a "behavior" analysis. Munetoh discloses a Finite State Machine (FSM) that processes the transmitted IML in order to determine the current state of the software running on the system. As Munetoh explains,

> … we have extended the format of the RM to incorporate the behavior model, allowing the PTS to use it when verifying the IRs. ***Intuitively, the behavior model is a finite state machine (FSM) that can consume an IML***. The PCR values and the IML in the IR are generated as a result of running the trusted platform, while the behavior model describes how they are measured, and how they should be verified.
>
> The FSM is a representation of the trusted-bootstrap and subsequent measurement processes of the platform. ***Each state in the behavior model represents a platform state, and each transition represents the integrity of the component that is loaded and executed***.
> (Ex. 1005 at p. 6, emphasis added)

243. The IML, which enables the behavior model of Munetoh, is also introspective. The '441 patent describes behavioral analysis as "introspective." It states, "The attestation broker 109 may request introspection based security context (for example, evaluations and ***behavioral*** or predictive analytics ***based on historic state information and measurements sampled over a period of time using a variety of inspection methods***)…" (Ex. 1001 at 9:37-46). The IML, from which the behavior model is derived, defines events and state transitions that occurred over the time of the system start-up. Munetoh provides an illustrative example:

> In this example, the digest must be 04df ... 2865 and 5a86 ... Id3e to make the transition from the initial state to statel, and state2 to state3, respectively. For the transition from statel to state2, only the event type 0x04 is required. Therefore, the IML in Fig. 5 passes the FSM.
> (Ex. 1005 at p. 7).

244. I noted with respect to limitation 1[d] that a POSITA would have been motivated to combine Munetoh with Rajan to draw on Rajan's dynamic execution analysis for fine-grained,

operation-level attestation. Rajan's trace-based analysis is also introspective because it analyzes the internals of the program service such as whether or not it is writing to a file.

> In this work, we focus on the trace-based requirements monitor… In a trace-based monitor, requirements are specified as logical formulas ***over program traces***. An example of such a trace-based requirement for the payment processing service could be, "***in no trace of the service implementation, the argument credit card number directly or indirectly flows to a persistence-related method, e.g., File.write(...)***."
> (Ex. 1006 at pp. 5-6, emphasis added).

245. This analysis disclosed by Rajan is dynamic because it is performed on the application as it is running.

246. Thus, I conclude that Munetoh in view of Rajan renders Claim 4 obvious to a POSITA.

F. Claim 5 is rendered obvious by Munetoh in view of Rajan

247. Claim 5 of the '441 patent is rendered obvious to a POSITA by Munetoh in view of Rajan. This claim, which depends from Claim 4, reads:

> *The method of claim 4, wherein the generating of the report indicating security risks associated with the application includes generating, by the attestation server, one or more security assertions that pertain to the received runtime execution context and the received introspective security context.*

248. As discussed above, Munetoh in view of Rajan discloses all limitations of Claims 1 and 4 from which Claim 5 depends.

249. Munetoh discloses "generating, by the attestation server, one or more security assertions that pertain to the received runtime execution context and the received introspective security context." As I explained with respect to limitation 1[e], Munetoh discloses generating vulnerability data based on the PCR and IML data, which constitute both the runtime execution

context and the introspective security context. (*See* Ex. 1005 at p. 6). The detection of vulnerabilities are security assertions that a specific file is vulnerable to a specific risk. As Munetoh explains:

> The PVA links the vulnerability information and the integrity information collected from Reference Manifests, and verifies the security state from the latest components. Vulnerability information in the XML format includes the CVE information stored in the database, and each CVE entry is associated with a security score for the CVSS.
>
> In the IMA the files are measured by using the SHA-1 hash function. Integrity information in an IR is composed of the hash values of the files measured with the IMA… ***In order to support file-based vulnerability detection, we extract the list of files in each package, and associate the vulnerability information of each package to each file.***
> (Ex. 1005 at p. 7, emphasis added).

250. As I explained with respect to limitation 1[e], the claimed "report" could be either the verification result (VR) or, alternatively, the vulnerability report, "which describes the presence of vulnerabilities and the degrees of security risks". (Ex. 1005 at p. 7). In either case, Munetoh is clear that generating such report involves generating security assertions.

251. Furthermore, I explained with respect to limitation 1[e] that a POSITA would have been motivated to use Munetoh in combination with Rajan to obtain both Rajan's analysis of dynamic vulnerabilities observed while the application is running ("runtime execution context") and Munetoh's analysis of static vulnerabilities determined from analyzing files on disk ("introspective security context"). Such combined analysis would produce the claimed "one or more security assertions that pertain to the received runtime execution context and the received introspective security context."

252. Thus, Munetoh in view of Rajan renders Claim 5 obvious to a POSITA.

G. Claim 6 is rendered obvious by Munetoh in view of Rajan

253. Claim 6 of the ’441 is rendered obvious to a POSITA by Munetoh in view of Rajan. This claim, which depends from Claim 1, reads:

> *The method according to claim 1, further comprising authenticating the application using a plurality of collaboration services*

254. As discussed above, Munetoh in view of Rajan discloses all limitations of Claim 1, from which Claim 6 depends.

255. Munetoh discloses authenticating the application using a plurality of collaboration services. Munetoh discloses that various software vendors (e.g., for the operating system as well as the applications) produce reference manifests (RMs) for the software they produce. “The vendors release software with RMs and the software is deployed into the target platforms.” (Ex. 1005 at p. 5). This is also illustrated in Figure 1 reproduced hereinbelow:

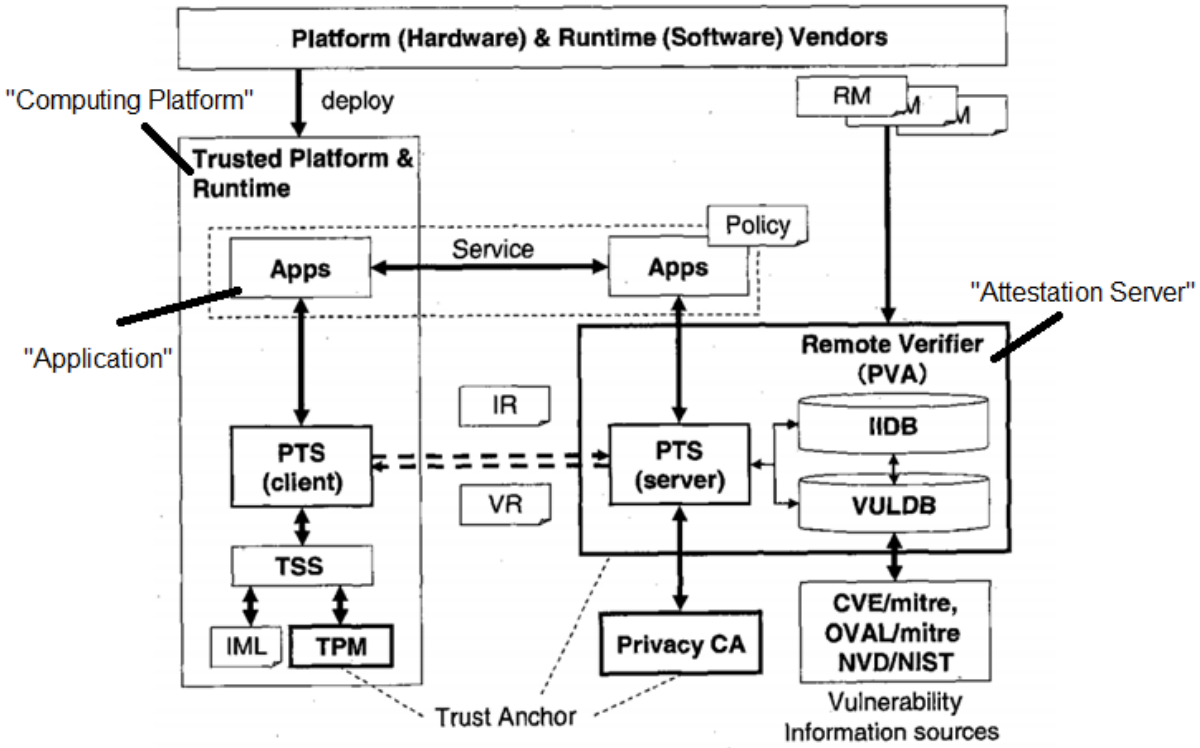


Fig. 1 PTS Client/Server architecture.

(Ex. 1005 at Figure 1, annotated).

256. The RMs are used to authenticate the application by integrity metrics. "The assertion made by a manufacturer on the composition of components and their integrity metrics (i.e., hash values) is called the Reference Manifest (RM). The RM is later compared with the runtime measurements to verify the integrity of the platform configuration." (Ex. 1005 at p. 4).

257. The vulnerability data is also produced through the collaboration of vendors. "The VULDB is the central repository of the known vulnerabilities, by aggregating information from vendors and security advisories." (Ex. 1005 at p. 5). Vulnerability data is also used to authenticate the application by vulnerability measures.

258. Thus, Munetoh in view of Rajan renders Claim 6 obvious to a POSITA.

H. Claim 7 is rendered obvious by Munetoh in view of Rajan

259. Claim 7 of the '441 is rendered obvious to a POSITA by Munetoh in view of Rajan. This claim, which depends from Claim 1, reads:

> *The method according to claim 1, further comprising controlling a user's transaction with the application by applying a set of authorization rules in accordance with the attestation results.*

260. As discussed above, Munetoh in view of Rajan discloses all limitations of Claim 1, from which Claim 7 depends.

261. Rajan discloses controlling a user's transaction with the application by applying a set of authorization rules in accordance with the attestation result. As discussed regarding limitations 1[e] and 1[f], Rajan discloses transmitting an attestation result in the form of approval or a notice of a violation. Rajan further discloses that the client limits the user's transaction (i.e., with the service) by aborting the operation if a violation is reported. See, for example, Procedure 5 from Rajan:

Procedure 5. Bind step for client's end.

```
1.  initTransaction();
2.  requestCert(brokerAddr,providerAddr,Id);
3.  sendRequest(providerAddr, params);
4.  result = receiveCert(brokerAddr,Id);
5.  if result == violationNotification OR timeOut then
6.    abortTransaction();
7.  else
8.    commitTransaction();
9.  end if
10. return;
```

(Ex. 1006 at p. 8).

262. Thus, Munetoh in view of Rajan renders Claim 7 obvious to a POSITA.

I. Claim 9 is rendered obvious by Munetoh in view of Rajan

263. Claim 9 of the ’441 is rendered obvious to a POSITA by Munetoh in view of Rajan. This claim, which depends from Claim 1, reads:

> *The method according to claim 1, further comprising providing confidence metrics in the attestation results indicating a level of security risk by different classifications such that a restriction on a user’s transaction with the application are applied based on the level of security risk indicated by the confidence metrics in the attestation results.*

264. As discussed above, Munetoh in view of Rajan discloses all limitations of Claim 1, from which Claim 9 depends.

265. Munetoh discloses providing confidence metrics in the attestation results indicating a level of security risk. Munetoh discloses that each CVE vulnerability is associated with a score or rating of the severity.

> The CVSS (Common Vulnerability Scoring System) [26] provides a method for rating vulnerability scores in an open and universal manner. This method evaluates vulnerabilities by three metrics such as base metrics, temporal metrics and environmental metrics. The NVD (National Vulnerability Database) [18] provides CVSS scores for most of the CVE entries, and thus each security advisory can be associated with a vulnerability score.
> (Ex. 1005 at p. 7).

266. Munetoh further discloses that the vulnerability result includes the scores. “Vulnerability information in the XML format includes the CVE information stored in the database, and each CVE entry is associated with a security score for the CVSS.” (Ex. 1005 at p. 8).

267. Munetoh also discloses security policies that prevent access to systems based on CVSS score as shown in Table 1, reproduced with annotations hereinbelow:

Table 1 Example of security properties.

Properties	Values	Comments
Transitive Trust		
CRTM Type	SRTM DRTM both	Platform w/ TPM v1.1b or v1.2 Platform w/ TPM v1.2 and CPU, chipset to support DRTM both BIOS and VMM must be valid
Range	BIOS Bootloader VMM OS(IMA)	The range of Transitive Trust is depend on the runtime implementation.
Integrity	Black list* White list	Check marked components only Does not allow unknown components
Vulnerability		
Validation type	Deny all Severity	Do not allow any vulnerable components e.g. decided by CVSS score
Identity		
Credential	Issuer	AIK credential should be issued by PrivacyCA, but other TTP could by accepted by policy
Manifest	Issuer	issued by vendor or PVA

(Ex. 1005 at p. 8, annotated).

268. Thus, Munetoh in view of Rajan renders Claim 9 obvious to a POSITA.

X. GROUND 2: CLAIMS 1-7 AND 9 ARE RENDERED OBVIOUS BY RAJAN IN VIEW OF MUNETOH

269. Claims 1-7 and 9 of the '441 patent are rendered obvious to a POSITA by Rajan in view of Munetoh, as explained in the sections below. I also explain that it would have been obvious for a POSITA to combine Rajan and Munetoh.

A. Motivation to Combine Rajan and Munetoh

270. A POSITA would have been motivated to combine Rajan and Munetoh. As discussed in Sections VIII.A and VIII.B., both Rajan and Munetoh are based on Remote

Attestation using TCG's specification. The core attestation technology, such as the use of TPMs and PCRs, is the same. A POSITA would have been motivated to combine Munetoh's static vulnerability analysis with Ranjan's dynamic trace-based analysis; Munetoh's static analysis enables policy decisions based on known security-risks associated with an application.

271. Furthermore, as I explained in Section IX.A., a POSITA would have found Rajan and Munetoh to be compatible and would have expected a high likelihood of success with minimal experimentation, given such compatibility and the similarity of their core technologies. I incorporate my analysis from Section IX.A. here by reference.

B. Claim 1 is rendered obvious by Rajan in view of Munetoh

272. Each of the limitations of Claim 1 of the '441 patent is rendered obvious to a POSITA by Rajan in view of Munetoh as described below.

1. Limitation 1[a]

273. Limitation 1[a] of the '441 patent reads:

> *A method of providing an attestation service for an application at runtime executing on a computing platform using an attestation server, comprising:*

274. I understand that limitation 1[a] is a "preamble" and that the preamble of claims is often not limiting. To the extent that limitation 1[a] is considered limiting, Rajan discloses it.

275. I incorporate my analysis from Section IX.B.1. herein by reference. The following Figure 1 (annotated) illustrates key elements of that analysis:

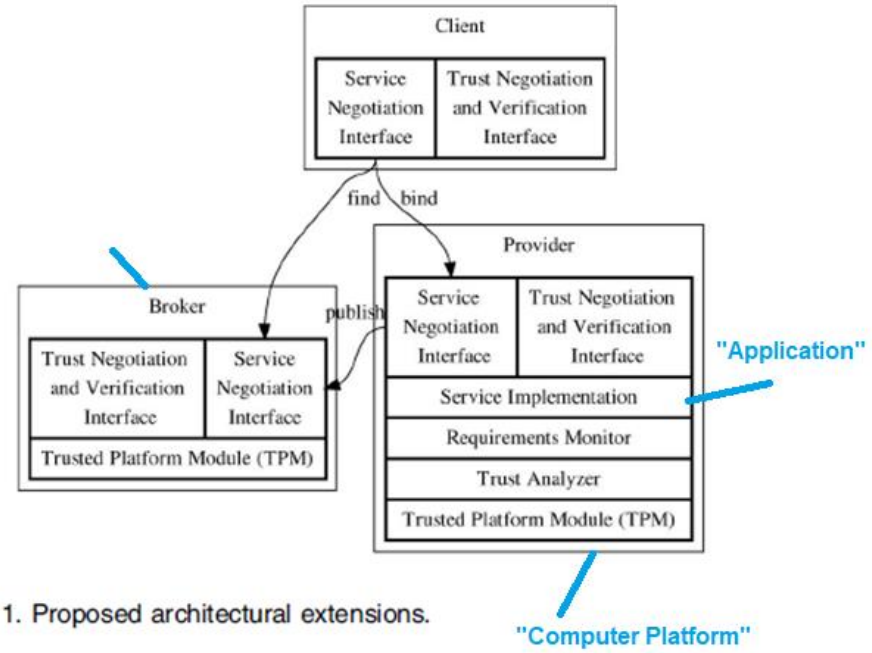


Fig. 1. Proposed architectural extensions.

276. Thus, I conclude that Rajan discloses limitation 1[a].

2. Limitation 1[b]

277. Limitation 1[b] of the ’441 patent reads:

receiving, by the attestation server remote from the computing platform:

278. Rajan discloses limitation 1[b]. I incorporate my analysis from Section IX.B.2. herein by reference.

3. Limitation 1[c]

279. Limitation 1[c] of the ’441 patent reads:

a runtime execution context indicating attributes of the application at runtime, wherein the attributes comprise one or more executable file binaries of the application and loaded components of the application; and

280. Rajan also discloses limitation 1[c]. I incorporate my analysis from Section IX.B.3. herein by reference.

281. As I explained in that section, Rajan discloses transmitting a static checksum as part of the integrity check. This checksum is one of two components of the integrity measurements provided by Rajan from the Service to the Broker. It is an attestation of the "software stack" including all of the low-level components, plus the "requirements monitor."

> During service execution, the broker and the provider must communicate to verify that: ***the provider and the broker are indeed the same principals that agreed to the contract during the publish step, the requirements monitor as well as the software stack on the provider's end is not compromised***, and the trace provided by this monitor does not violate the properties desired by the client.
>
> The trust analyzer component, which derives its root of trust from the TPM on the provider's side, serves to verify the integrity of the requirements monitor. ***It provides a report that includes an immutable integrity measurement of the local architecture along with a time stamp to indicate the freshness of the measurement***.
> (Ex. 1006 at p. 6, emphasis added)
>
> This trace, along with static checksum of the software stack, is sent to the trusted third party [broker] (step 7). ***To remind the reader, the static checksum is created by computing the SHA1 hash of the software stack up to the requirements monitor*** and is essential to validate the current state of the service implementation (step 5). This hash is signed using the AIK on the TPM of the service provider (step 6).
> (Ex. 1006 at p. 7, emphasis added).

282. The requirements monitor, now trusted by attestation, is then subsequently used to analyze the performance of the service software at runtime.

283. Thus, I conclude that Rajan discloses limitation 1[c].

4. Limitation 1[d]

284. Limitation 1[d] of the '441 patent reads:

> *a security context providing security information about the application, wherein the security information comprises an execution analysis of the one or more executable file binaries and the loaded components;*

285. Rajan in view of Munetoh discloses limitation 1[d].

286. As I discussed in Ground 1 with respect to this limitation, Rajan discloses analyzing runtime behavior of an application to analyze security information about the application. I incorporate my analysis from Section IX.V.4. herein by reference.

287. As also discussed in Ground 1, Munetoh discloses transmitting additional security information about the application. For example, Munetoh discloses transmitting the Integrity Measurement Log (IML) that "which includes a list of components loaded on the platform and measured by the TPM, and how these components are composed." (Ex. 1005 at p. 3). A POSITA would have been motivated to combine this security contextual information into Rajan so that Rajan's broker, which verifies the attestation data, would be able to know which components are loaded and how they are composed.

288. Thus, Rajan in view of Munetoh discloses limitation 1[d].

5. Limitation 1[e]

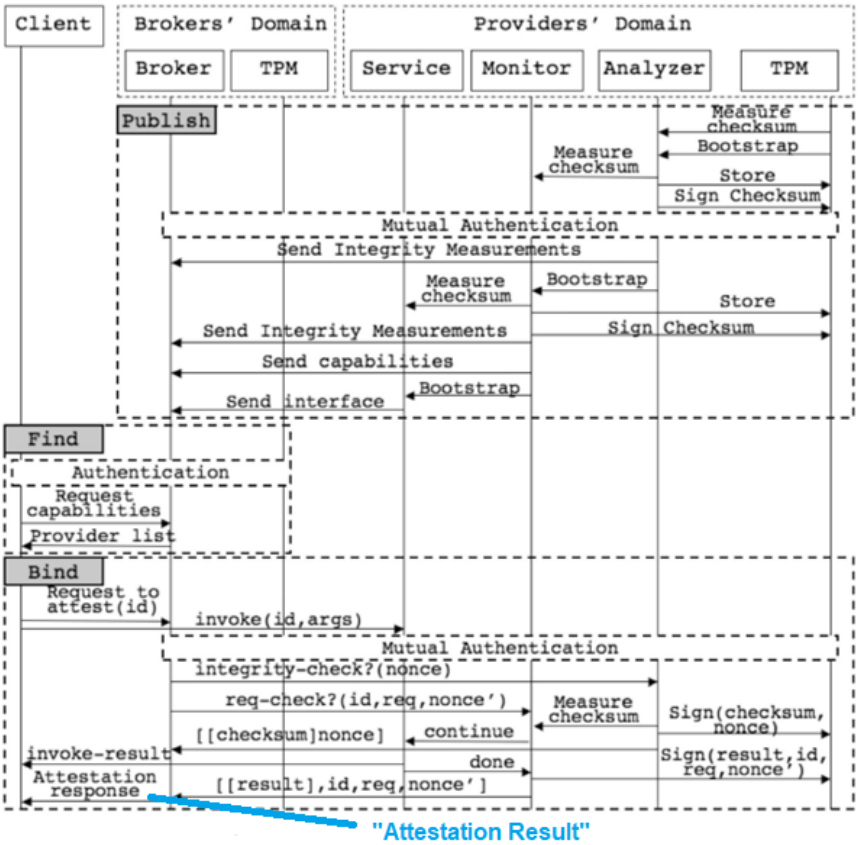
289. Limitation 1[e] of the '441 patent reads:

> *generating, by the attestation server, a report indicating security risks associated with the application based on the received runtime execution context and the received security context, as an attestation result; and*

290. Rajan in view of Munetoh discloses limitation 1[e].

291. As I discussed in Ground 1 with respect to this limitation, Rajan conducts analysis of dynamic vulnerabilities observed while the application is running and Munetoh conducts analysis of static vulnerabilities determined from analyzing files on disk. As I explained in the Ground 1 analysis, a POSITA would have been motivated to get both static and dynamic analyses of applications. A POSITA would have understood that static analyses enable the system to reject

an unsafe application even before it starts processing while the dynamic analysis enforces context-specific behavior (e.g., an activity that might be safe sometimes but not at others). Figure 2 of Rajan, reproduced herein below (annotated) illustrates some of these concepts.



292. For details about how Rajan and Munetoh conduct these analyses, I incorporate Section IX.B.5. herein by reference.

293. Thus, I conclude that Rajan in view of Munetoh discloses limitation 1[e].

6. Limitation 1[f]

294. Limitation 1[f] of the '441 patent reads:

sending, by the attestation server, the attestation result associated with the application.

295. Rajan discloses limitation 1[f].

296. As I discussed in Ground 1 with respect to this limitation, Rajan and Munetoh both disclose sending results of their analyses. For details about Rajan's disclosures with respect to this limitation, I incorporate Section IX.B.6 herein by reference.

297. Thus, I conclude that Rajan discloses limitation 1[f].

298. Thus, I conclude that Rajan in view of Munetoh renders Claim 1 obvious to a POSITA.

C. Claim 2 is rendered obvious by Rajan in view of Munetoh

299. Claim 2, which depends from Claim 1, is rendered obvious to a POSITA by Rajan in view of Munetoh.

1. Limitation 2[a]

300. Limitation 2[a] reads:

> *The method of claim 1, further comprising:*
>
> *generating, by the attestation server, an application artifact as a reference for changes in a subsequent execution context; and*

301. Rajan in view of Munetoh discloses limitation 2[a]. As discussed above, Rajan in view of Munetoh discloses all limitations of Claim 1, from which Claim 2 depends.

302. I conclude that Rajan in view of Munetoh discloses limitation 2[a], consistent with my analysis from Section IX.C.1, which I incorporate herein by reference.

2. Limitation 2[b]

303. Limitation 2[b] reads:

> *sending the generated application artifact such that subsequent changes to the runtime execution context are tracked based on the generated application artifact.*

304. I conclude that Rajan in view of Munetoh discloses limitation 2[b], consistent with my analysis from Section IX.C.2., which I incorporate herein by reference.

305. Thus, I conclude that Rajan in view of Munetoh renders Claim 2 obvious to a POSITA.

D. Claim 3 is rendered obvious by Rajan in view of Munetoh

306. Claim 3 of the '441 patent is rendered obvious to a POSITA by Rajan in view of Munetoh. This claim, which depends from Claim 2, reads:

> *The method according to claim 2, further comprising digitally signing the attestation results, wherein the attributes further comprise parent-child process associations of the application.*

307. As discussed above, Rajan in view of Munetoh discloses all limitations of Claims 1 and 2, from which Claim 3 depends.

308. As I explained with respect to Ground 1 for this claim, Munetoh discloses both the digital signing through HTTPS transmissions and the parent-child process associations through the composition analysis of components. A POSITA would have been motivated to incorporate both teachings of Munetoh into Rajan's teachings. With respect to HTTPS, Rajan notes that

> Our work is orthogonal and complementary to the secrecy and authenticity research in the Web services security community. We do not focus on securing the interaction between service providers, brokers, and clients, which has been the main focus of many existing approaches, e.g., current standards such as WS-Security [16] and WS-Trust [17] or proposals such as that by Skogsurd et al. [18].
> (Ex. 1006 at p. 2).

309. Munetoh notes that HTTPS is an alternative to WS-Trust. (Ex. 1005 at p. 6; *see also, e.g.*, Ex. 1008 *generally*). Thus, a POSTIA would have been motivated to combine Munetoh's HTTPS communications with Rajan's disclosures as they are "complementary," as explained above.

310. Moreover, I have already explained with respect to limitation 1[d] of this Ground 2 that a POSITA would have been motivated to combine Munetoh's IML security data into Rajan's teachings. In my analysis of this limitation with respect to Ground 1, I explained that Munetoh's IML data permits determining the "composition" of components, which enables a determination of parent-child process associations.

311. Additional details of how Munetoh discloses this claim is set forth in my analysis of this claim in Ground 1, which I incorporate by reference.

312. Thus, Rajan in view of Munetoh renders Claim 3 obvious to a POSITA.

E. Claim 4 is rendered obvious by Rajan in view of Munetoh

313. Claim 4 of the '441 patent is rendered obvious to a POSITA by Rajan in view of Munetoh. This claim, which depends from Claim 1, reads:

> *The method of claim 1, wherein the received security context is an introspective security context, and wherein the execution analysis is a static, dynamic, or virtual analysis of the one or more executable file binaries and the loaded components.*

314. As discussed above, Rajan in view of Munetoh discloses all limitations of Claim 1, from which Claim 4 depends.

315. I conclude that Rajan in view of Munetoh discloses claim 3, consistent with my analysis from Section IX.E., which I incorporate herein by reference.

316. Thus, I conclude that Rajan in view of Munetoh renders Claim 4 obvious to a POSITA.

F. Claim 5 is rendered obvious by Rajan in view of Munetoh

317. Claim 5 of the '441 patent is rendered obvious to a POSITA by Rajan in view of Munetoh. This claim, which depends from Claim 4, reads:

The method of claim 4, wherein the generating of the report indicating security risks associated with the application includes generating, by the attestation server, one or more security assertions that pertain to the received runtime execution context and the received introspective security context.

318. As discussed above, Rajan in view of Munetoh discloses all limitations of Claims 1 and 4 from which Claim 5 depends.

319. I conclude that Rajan in view of Munetoh discloses claim 5, consistent with my analysis from Section IX.F., which I incorporate herein by reference.

320. Thus, Rajan in view of Munetoh renders Claim 5 obvious to a POSITA.

G. Claim 6 is rendered obvious by Rajan in view of Munetoh

321. Claim 6 of the '441 is rendered obvious to a POSITA by Rajan in view of Munetoh. This claim, which depends from Claim 1, reads:

The method according to claim 1, further comprising authenticating the application using a plurality of collaboration services

322. As discussed above, Rajan in view of Munetoh discloses all limitations of Claim 1, from which Claim 6 depends.

323. I have already explained that a POSITA would have been motivated to incorporate the static application analysis of Munetoh into Rajan's teachings in Section X.A. I incorporate my analysis of that section by reference. As set forth in my analysis of this claim in Ground 1 in Section IX.G., Munetoh also discloses that the static analysis made use of external services, which are "collaboration services," for creating and maintaining both the vulnerability databases and the reference manifests for applications. I incorporate this analysis from Section IX.G. into this section by reference.

324. Thus, Rajan in view of Munetoh renders Claim 6 obvious to a POSITA.

H. Claim 7 is rendered obvious by Rajan in view of Munetoh

325. Claim 7 of the ’441 is rendered obvious to a POSITA by Rajan in view of Munetoh. This claim, which depends from Claim 1, reads:

> *The method according to claim 1, further comprising controlling a user’s transaction with the application by applying a set of authorization rules in accordance with the attestation results.*

326. As discussed above, Rajan in view of Munetoh discloses all limitations of Claim 1, from which Claim 7 depends. I further conclude that Rajan in view of Munetoh renders Claim 7 obvious to a POSITA, consistent with my analysis from Section IX.H., which I incorporate herein by reference.

I. Claim 9 is rendered obvious by Rajan in view of Munetoh

327. Claim 9 of the ’441 is rendered obvious to a POSITA by Rajan in view of Munetoh. This claim, which depends from Claim 1, reads:

> *The method according to claim 1, further comprising providing confidence metrics in the attestation results indicating a level of security risk by different classifications such that a restriction on a user’s transaction with the application are applied based on the level of security risk indicated by the confidence metrics in the attestation results.*

328. As discussed above, Rajan in view of Munetoh discloses all limitations of Claim 1, from which Claim 9 depends.

329. I have already explained that a POSITA would have been motivated to incorporate the static application analysis of Munetoh into Rajan’s teachings in Section X.A. I incorporate my analysis of that section by reference. As set forth in my analysis of this claim in Ground 1 in

Section IX.I., Munetoh also discloses that the static analysis made use of vulnerability scoring systems such as CVSS and that the policies applied to applications were tied to these scores. I incorporate this analysis from Section IX.I. into this section by reference.

330. Thus, Rajan in view of Munetoh renders Claim 9 obvious to a POSITA.

XI. THERE ARE NO OBJECTIVE INDICIA OF NONOBVIOUSNESS

331. I am not aware of any objective indicia of non-obviousness related specifically to the ’441 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 ’441 patent.

XII. CONCLUSION

332. 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 6 April, 2023

By: _____
Seth James Nielson