UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CROWDSTRIKE, INC.
Petitioner

v.

TAASERA
Patent Owner

Case No. IPR2024-00027
Patent No. 7,673,137

DECLARATION OF DR. MARKUS JAKOBSSON, PH.D.

TABLE OF CONTENTS

I, Dr. Markus Jakobsson, declare the following:

I. INTRODUCTION

1. I have been retained by counsel for Petitioner as a technical expert in the above-captioned case. Specifically, I have been asked to render certain opinions in regard to the IPR petition with respect to U.S. Patent No. 7,673,137 (the "'137 Patent"). I understand that the Challenged Claims are claims 1, 2, 6, 8, 9, 12, and 25, and my opinions are limited to those claims. My compensation in this matter is not based on the substance of my opinions or the outcome of this matter. I have no financial interest in Petitioner.

A. Materials Reviewed

2. In reaching my opinions in this matter, I have reviewed the following materials:

- Exhibit 1001 – U.S. Patent No. 7,673,137 ("'137 Patent");
- Exhibit 1002 – File History of U.S. Patent No. 7,673,137 ("'137 Patent File History");
- Exhibit 1005 – Asit Dan, Ajay Mohindra, Rajiv Ramaswami and Dinkar Sitaram, *ChakraVyuja (CV): A Sandbox Operating System Environment for Controlled Execution of Alien Code*, IBM Research Division ("Dan");
- Exhibit 1006 – U.S. Patent Publication No. 2002/0099952 to Lambert et al. ("Lambert");
- Exhibit 1007 – U.S. Patent No. 7,085,928 to Schmid et al. ("Schmid");
- Exhibit 1013 – Webster's New World Dictionary of Computer Terms, Simon & Schuster, Inc. (5th ed., 1994) ("Dictionary");

- Exhibit 1014 – *Intel Architecture Software Developer's Manual Volume 3: System Programming*, Intel Corporation (1999) ("Intel System Programming Manual");

- Exhibit 1015 – David A. Solomon, *The Windows NT Kernel Architecture*, 31 Computer 40-47 (October 1998) ("Solomon");

- Exhibit 1016 – Carol Neshevich, *NT 5.0 becomes Windows 2000*, 8 New World Canada 6, https://www.proquest.com/openview/fa3551c04de4b0e4a14ccee74eb12f0f/1?pq-origsite=gscholar&cbl=43820 (November 20, 1998) ("Neshevich");

- Exhibit 1017 – *Microsoft Releases Windows 2000 to Manufacturing*, Microsoft Corporation (December 15, 1999) https://news.microsoft.com/1999/12/15/microsoft-releases-windows-2000-to-manufacturing/ ("Microsoft Windows 2000 Press Release");

- Exhibit 1018 – Terrance Mitchem, Raymond Lu, Richard O'Brien, Kent Larson, *Linux Kernel Loadable Wrappers*, in Proceedings DARPA Information Survivability Conference and Exposition (IEEE, January 2000) ("Mitchem");

- Exhibit 1019 – Microsoft Press Computer Dictionary, Microsoft Corporation (3rd ed., 1997) ("Microsoft Computer Dictionary");

- Exhibit 1020 – Gary Wiggins, *Living with MalWare*, Security Essentials version 1.2d, (SANS Institute, 2001) ("Wiggins");

- Exhibit 1021 – Saliman Manap, *Rootkit: Attacker undercover tools*., Global Information Assurance Certification Paper, (10/21/2001) ("Manap");

- Exhibit 1022 – Massimo Bernaschi, Emanuele Gabrielli, Luigi V. Mancini, *Operating System Enhancement to Prevent Misuse of System Calls*, ACM CCS'00 (2000) ("Bernaschi");

- Exhibit 1023 – Pietro Iglio, *TrustedBox: a Kernel-Level Integrity Checker*, in Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99) (IEEE, December 1999) ("Iglio");

- Exhibit 1024 – F. De Paoli, A. L. Dos Santos, R. A. Kemmerer, *Vulnerability of "Secure" Web Browsers*, In Proceedings of the National Information Systems Security Conference (May 26, 1997) ("De Paoli");

- Exhibit 1025 – Bart Preneel, *Cryptographic Hash Functions*, 5 European Transactions on Telecommunications (1994) ("Preneel");

- Exhibit 1026 – karlrupp, *Microprocessor Trend Data*, GitHub, https://github.com/karlrupp/microprocessor-trend-data#readme (Feb 22, 2022) ("Microprocessor Trend Data"); and

- Exhibit 1027 –*Why is Sandboxing important?,* STL Tech https://stl.tech/blog/what-is-sandboxing-and-why-do-we-need-it/ (October 14, 2022) ("STL Tech").

B. Background and Qualifications

3. I have summarized in this section my educational background, career history, and other qualifications relevant to this matter. I have also included a current version of my curriculum vitae, which is attached as Appendix A.

4. I received a Master of Science degree in Computer Engineering from the Lund Institute of Technology in Sweden in 1993, a Master of Science degree in Computer Science from the University of California at San Diego in 1994, and a Ph.D. in Computer Science from the University of California at San Diego in 1997, specializing in Cryptography. During and after my Ph.D. studies, I was also a researcher at the San Diego Supercomputer Center and General Atomics, where I did research on authentication, privacy, and security. Much of my work was on distributed protocols to enable security and privacy, and my Ph.D. thesis described a distributed electronic payment system.

5. Since earning my doctorate degree over twenty-six years ago, I have been an employee, entrepreneur, and consultant in the computer systems, security, and the anti-fraud industry. I have worked extensively with technologies related to computer security, mobile security, phishing, malware detection, quantitative and qualitative fraud analysis, and disruptive security. I have worked as an engineer, researcher or technical advisor at a number of leading companies in the computer industry, including PayPal, Qualcomm, Bell Labs, and LifeLock. I have also taught and performed research at several prestigious universities, including New York University (NYU) and Indiana University (IU). Further, I have been materially involved in designing, developing, testing, and assessing technologies for computer and network security, digital rights management, trust establishment, randomness, cryptography, socio-technical fraud, malware detection, detection of malicious emails, user interfaces, authentication, and fraud detection for over twenty years.

6. From 1997 to 2001, I was a Member of Technical Staff at Bell Labs, where I did research on authentication, privacy, multi-party computation, contract exchange, digital commerce including crypto payments, and fraud detection and prevention. An example publication of mine from this time period includes "Secure distributed computation in cryptographic applications" (U.S. Patent No. 6,950,937, 2001 priority date.) At the end of my tenure at Bell Labs, I had started to research

identity theft years before the banking industry became aware of the existence of phishing attacks.

7. From 2001 to 2004, I was a Principal Research Scientist at RSA Labs, where I worked on predicting future fraud scenarios in commerce and authentication and developed solutions to those problems. Much of my work related to the development of anti-fraud mechanisms, addressing authentication abuse and identity theft. During that time, I predicted the rise of what later became known as phishing. I also laid the groundwork for what I termed "proof of work," and described how this could be applied in the context of distributed computations such as mining of crypto payments; this work later came to influence the development of BitCoin. I was also an Adjunct Associate Professor in the Computer Science department at New York University from 2002 to 2004, where I taught cryptographic protocols, with an emphasis on authentication.

8. From 2004 to 2016, I held a faculty position at Indiana University at Bloomington, first as an Associate Professor of Computer Science, Associate Professor of Informatics, Associate Professor of Cognitive Science, and Associate Director of the Center for Applied Cybersecurity Research (CACR) from 2004 to 2008; and then as an Adjunct Associate Professor from 2008 to 2016. I was the most senior security researcher at Indiana University, where I built a research group focused on online fraud and countermeasures, resulting in over 50 publications and

two books. One of these books, "Crimeware: Understanding New Attacks and Defenses" (Wiley, 2008), described malware-based attacks, online abuse in general, and countermeasures to attacks and abuses of these types.

9. While a professor at Indiana University, I was also employed by Xerox PARC, PayPal, and Qualcomm to provide thought leadership to their security groups. I was a Principal Scientist at Xerox PARC from 2008 to 2010, a Director and Principal Scientist of Consumer Security at PayPal from 2010 to 2013, a Senior Director at Qualcomm from 2013 to 2015, Chief Scientist at Agari from 2016 to 2018, and Chief of Security and Data Analytics at Amber Solutions from 2018 to 2020.

10. I was hired by Qualcomm as they acquired a startup I founded, FatSkunk. FatSkunk had developed algorithms for retroactive detection of malware, with applications to low-power platforms such as mobile devices. Example publications include "Retroactive Detection of Malware With Applications to Mobile Platforms" by Markus Jakobsson and Karl-Anders Johansson, "Practical and Secure Software-Based Attestation" by Markus Jakobsson and Karl-Anders Johansson, and U.S. Patent 8,370,935, titled "Auditing a Device". My work at Qualcomm related to authentication of users and detection of malware, and the integration of the FatSkunk technology in Qualcomm products.

11. Agari is a cybersecurity company that develops and commercializes technology to protect enterprises, their partners and customers from advanced email phishing attacks. At Agari, my research studied and addressed trends in online fraud, especially as related to email, including problems such as Business Email Compromise, Ransomware, and other abuses based on social engineering and identity deception. My work primarily involved identifying trends in fraud and computing before they affected the market, and developing and testing countermeasures, including technological countermeasures, user interaction and education. Example publications include "Detecting computer security risk based on previously observed communications" (U.S. Patent No. 10,715,543 B2, 2016 priority date) and "Using message context to evaluate security of requested data" (U.S. Patent No. 10,805,314 B2, 2017 priority date).

12. In 2020, after leaving Agari, I was the Chief of Security and Data Analytics at Amber Solutions, an IoT startup in the San Francisco Bay Area. Amber Solutions is a cybersecurity company that develops home and office automation technologies. At Amber Solutions, my research addressed privacy, user interfaces and authentication techniques in the context of ubiquitous and wearable computing. My work often related to security in constrained computing environments, as exemplified in "Configuration and management of smart nodes with limited user interfaces" (U.S. Patent No. 10,887,447).

13. I left Amber Solutions to take the role of Chief Scientist at ByteDance in 2019, where I guided the security research both for ByteDance and TikTok until I left to co-found Artema Labs, where I am the Chief Scientist. My work at ByteDance involved identifying new threats, developing new solutions, and guiding research in areas relating to cryptography and Internet security.

14. I am also the Chief Technology Officer at ZapFraud, a cybersecurity company that develops techniques to detect deceptive emails, such as Business Email Compromise ("BEC") emails. At ZapFraud, my research studies and addresses computer abuse, including social engineering, malware and privacy intrusions, and has also involved studying security aspects related to cloud computing. My work primarily involves identifying risks, developing protocols and user experiences, and evaluating the security of proposed approaches.

15. I also work as an independent security researcher and consultant in the fields of computer/mobile security, fraud detection/prevention, digital rights management, malware, phishing, crimeware, mobile malware, and cryptographic protocols. In addition, I am a visiting research fellow of the Anti-Phishing Working Group, an international consortium focused on unifying the global response to cybercrime in the public and private sectors.

16. I have also served on the advisory board of several companies, including Metaforic, a company that developed technology for watermarking

software and provided security software with a built-in framework that resists attacks from malware, bots, hackers and other attempts. Metaforic was acquired by Inside Secure in 2014.

17. I have founded or co-founded several successful computer security companies. In 2005, I co-founded RavenWhite Security, a provider of authentication solutions, and I am currently its Chief Technical Officer. RavenWhite owns intellectual property related to authentication, such as "Performing authentication" (U.S. Patent No. 10,079,823 2007 priority date). In 2007 I co-founded Extricatus, one of the first companies to address consumer security education. In 2009 I founded FatSkunk, a provider of mobile malware detection software; I served as Chief Technical Officer of FatSkunk from 2009 to 2013, when FatSkunk was acquired by Qualcomm, and I became a Qualcomm employee. FatSkunk developed anti-virus technology. In 2013 I founded ZapFraud—the same company described above, where I currently serve as Chief Technical Officer. In 2014 I co-founded RightQuestion, a security consulting company.

18. I have additionally served as a member of the fraud advisory board at LifeLock (an identity theft protection company); a member of the technical advisory board at CellFony (a mobile security company); a member of the technical advisory board at PopGiro (a user reputation company); a member of the technical advisory board at MobiSocial dba Omlet (a social networking company); and a member of

the technical advisory board at Stealth Security (an anti-fraud company). I have provided anti-fraud consulting to KommuneData (a Danish government entity), J.P. Morgan Chase, PayPal, Boku, and Western Union.

19. I have authored seven books and over 100 peer-reviewed publications and have been a named inventor on over 200 patents and patent applications. A large number of these relate to Internet security, authentication, malware, spoofing and related topics.

20. Examples of textbooks I have authored or co-authored on the topic of Internet security include:

- "Crimeware: Understanding New Attacks and Defenses" (Symantec Press, 2008)
- "Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft" (Wiley, 2006);
- "The Death of the Internet" (Wiley, 2012);
- "Understanding Social Engineering Based Scams" (Springer 2016);
- "Towards Trustworthy Elections: New Directions in Electronic Voting" (Springer, 2010).

21. Books translated to Chinese and Korean have separate titles and years of release.

22. I have authored numerous publications regarding Internet security, including Internet security and malware detection. Relevant examples include:

- Tamper-Evident Digital Signatures: Protecting Certification Authorities Against Malware Jong Youl Choi, Philippe Golle, and Markus Jakobsson;
- "Remote Harm-Diagnostics", (PDF) Privacy-preserving history mining for web browsers (researchgate.net);
- "Server-Side Detection of Malware Infection", NSPW Proceedings on New Security Paradigms Workshop (2009).

23. An exemplary relevant patent publication includes U.S. Patent No. 8,578,174. This patent was assigned to Palo Alto Research Center Incorporated arising out of my employment.

24. My opinions are based on my years of education, research, and experience, as well as my study of relevant materials. In forming my opinions, I have also considered the materials identified in this declaration and in the Petition.

25. In sum, I have extensive experience both as a developer and a researcher relating to computer security.

II. LEGAL FRAMEWORK

26. I am a technical expert and do not offer any legal opinions. However, counsel has informed me as to certain legal principles regarding patentability and related matters under United States patent law, which I have applied in performing my analysis and arriving at my technical opinions in this matter.

27. I have been informed that claim terms are to be given the meaning they would have to a person having ordinary skill in the art at the time of the invention, taking into consideration the patent, its file history, and, secondarily, any applicable extrinsic evidence (e.g., dictionary definitions).

28. I have also been informed that the implicit or inherent disclosures of a prior art reference may anticipate the claimed invention. Specifically, if a person having ordinary skill in the art at the time of the invention would have known that the claimed subject matter is necessarily present in a prior art reference, then the prior art reference may "anticipate" the claim. Therefore, a claim is "anticipated" by the prior art if each and every limitation of the claim is found, either expressly or inherently, in a single item of prior art.

29. I have also been informed that a person cannot obtain a patent on an invention if the differences between the invention and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art. I have been informed that a conclusion of obviousness may be founded upon more than a single item of prior art. I have been further informed that obviousness is determined by evaluating the following factors: (1) the scope and content of the prior art, (2) the differences between the prior art and the claim at issue, (3) the level of ordinary skill in the pertinent art, and (4) secondary considerations of non-obviousness. In addition, the

obviousness inquiry should not be done in hindsight. Instead, the obviousness inquiry should be done through the eyes of a person having ordinary skill in the relevant art at the time the patent was filed.

30. In considering whether certain prior art renders a particular patent claim obvious, counsel has informed me that I can consider the scope and content of the prior art, including the fact that one of skill in the art would regularly look to the disclosures in patents, trade publications, journal articles, industry standards, product literature and documentation, texts describing competitive technologies, requests for comment published by standard setting organizations, and materials from industry conferences, as examples. I have been informed that for a prior art reference to be proper for use in an obviousness analysis, the reference must be "analogous art" to the claimed invention. I have been informed that a reference is analogous art to the claimed invention if: (1) the reference is from the same field of endeavor as the claimed invention (even if it addresses a different problem); or (2) the reference is reasonably pertinent to the problem faced by the inventor (even if it is not in the same field of endeavor as the claimed invention). In order for a reference to be "reasonably pertinent" to the problem, it must logically have commended itself to an inventor's attention in considering his problem. In determining whether a reference is reasonably pertinent, one should consider the problem faced by the inventor, as reflected either explicitly or implicitly, in the specification. I believe that

all of the references that my opinions in this IPR are based upon are well within the range of references a person having ordinary skill in the art would consult to address the type of problems described in the Challenged Claims.

31. I have been informed that, in order to establish that a claimed invention was obvious based on a combination of prior art elements, a clear articulation of the reason(s) why a claimed invention would have been obvious must be provided. Specifically, I am informed that a combination of multiple items of prior art renders a patent claim obvious when there was an apparent reason for one of ordinary skill in the art, at the time of the invention, to combine the prior art, which can include, but is not limited to, any of the following rationales: (A) combining prior art methods according to known methods to yield predictable results; (B) substituting one known element for another to obtain predictable results; (C) using a known technique to improve a similar device in the same way; (D) applying a known technique to a known device ready for improvement to yield predictable results; (E) trying a finite number of identified, predictable potential solutions, with a reasonable expectation of success; (F) identifying that known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to one of ordinary skill in the art; or (G) identifying an explicit teaching, suggestion, or motivation in

the prior art that would have led one of ordinary skill to modify the prior art reference or to combine the prior art references to arrive at the claimed invention.

32. I am informed that the existence of an explicit teaching, suggestion, or motivation to combine known elements of the prior art is a sufficient, but not a necessary, condition to a finding of obviousness. This so-called "teaching suggestion-motivation" test is not the exclusive test and is not to be applied rigidly in an obviousness analysis. In determining whether the subject matter of a patent claim is obvious, neither the particular motivation nor the avowed purpose of the patentee controls. Instead, the important consideration is the objective reach of the claim. In other words, if the claim extends to what is obvious, then the claim is invalid. I am further informed that the obviousness analysis often necessitates consideration of the interrelated teachings of multiple patents, the effects of demands known to the technological community or present in the marketplace, and the background knowledge possessed by a person having ordinary skill in the art. All of these issues may be considered to determine whether there was an apparent reason to combine the known elements in the fashion claimed by the patent.

33. I also am informed that in conducting an obviousness analysis, a precise teaching directed to the specific subject matter of the challenged claim need not be sought out because it is appropriate to take account of the inferences and creative steps that a person of ordinary skill in the art would employ. The prior art considered

can be directed to any need or problem known in the field of endeavor at the time of invention and can provide a reason for combining the elements of the prior art in the manner claimed. In other words, the prior art need not be directed towards solving the same specific problem as the problem addressed by the patent. Further, the individual prior art references themselves need not all be directed towards solving the same problem. I am informed that common sense is important and should be considered. Common sense teaches that familiar items may have obvious uses beyond their primary purposes.

34. I also am informed that the fact that a particular combination of prior art elements was "obvious to try" may indicate that the combination was obvious even if no one attempted the combination. If the combination was obvious to try (regardless of whether it was actually tried) or leads to anticipated success, then it is likely the result of ordinary skill and common sense rather than innovation. I am further informed that in many fields it may be that there is little discussion of obvious techniques or combinations, and it often may be the case that market demand, rather than scientific literature or knowledge, will drive the design of an invention. I am informed that an invention that is a combination of prior art must do more than yield predictable results to be non-obvious.

35. I am informed that for a patent claim to be obvious, the claim must be obvious to a person of ordinary skill in the art at the time of the invention. I am

informed that the factors to consider in determining the level of ordinary skill in the art include (1) the educational level and experience of people working in the field at the time the invention was made, (2) the types of problems faced in the art and the solutions found to those problems, and (3) the sophistication of the technology in the field.

36. I am informed that it is improper to combine references where the references teach away from their combination. I am informed that a reference may be said to teach away when a person of ordinary skill in the relevant art, upon reading the reference, would be discouraged from following the path set out in the reference, or would be led in a direction divergent from the path that was taken by the patent applicant. In general, a reference will teach away if it suggests that the line of development flowing from the reference's disclosure is unlikely to be productive of the result sought by the patentee. I am informed that a reference teaches away, for example, if (1) the combination would produce a seemingly inoperative device, or (2) the references leave the impression that the product would not have the property sought by the patentee. I also am informed, however, that a reference does not teach away if it merely expresses a general preference for an alternative invention but does not criticize, discredit, or otherwise discourage investigation into the invention claimed.

37. I am informed that the final determination of obviousness must also consider "secondary considerations" if presented. In most instances, the patentee raises these secondary considerations of non-obviousness. In that context, the patentee argues an invention would not have been obvious in view of these considerations, which include: (a) commercial success of a product due to the merits of the claimed invention; (b) a long-felt, but unsatisfied need for the invention; (c) failure of others to find the solution provided by the claimed invention; (d) deliberate copying of the invention by others; (e) unexpected results achieved by the invention; (f) praise of the invention by others skilled in the art; (g) lack of independent simultaneous invention within a comparatively short space of time; (h) teaching away from the invention in the prior art.

38. I am further informed that secondary considerations evidence is only relevant if the offering party establishes a connection, or nexus, between the evidence and the claimed invention. The nexus cannot be based on prior art features. The establishment of a nexus is a question of fact. While I understand that the Patent Owner here has not offered any secondary considerations at this time, I will supplement my opinions if the Patent Owner raises secondary considerations during this proceeding.

III. OPINION

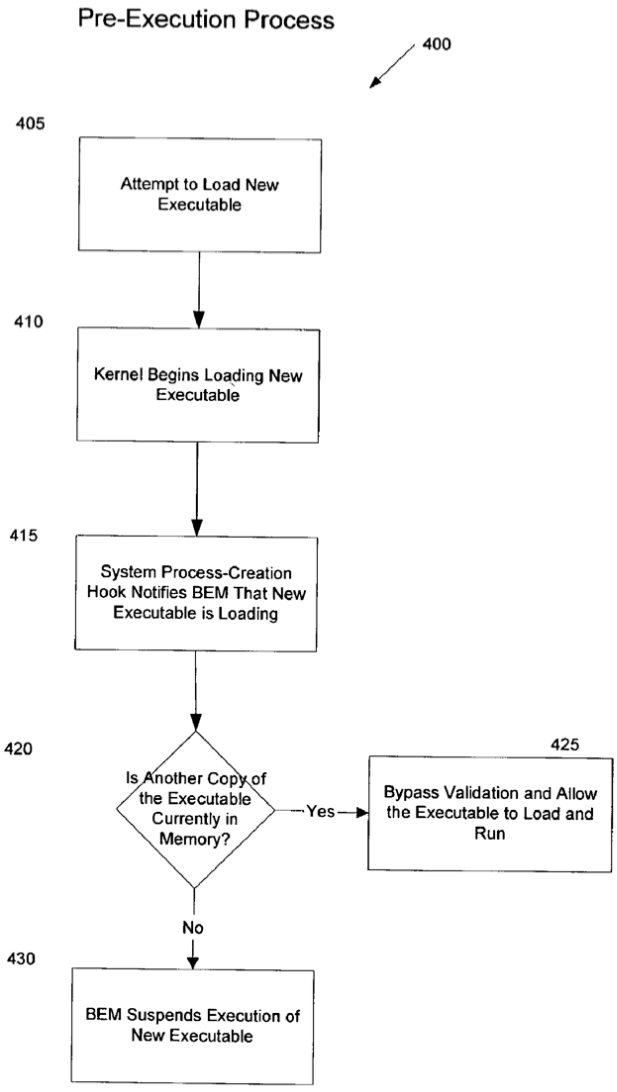
A. Overview of the '137 Patent

39. The ’137 Patent is directed to a protector system that includes “a two-step process to ensure that software programs do not perform malicious activities which may damage the computing device.” ’137 Patent, Abstract. The ’137 Patent describes a need for “early detection of security threats to a computing device or network before any harm can be done.” *Id*. at 3:1-8. “[T]here is a need in the art for a security system which will provide early detection of security threats to a computing device or network before any harm can be done.” *Id*. at 3:5-16.

40. To purportedly solve this problem, the ’137 Patent describes a protector system that uses “a two-phased approach at the kernel level of the computing device’s operating system” to analyze and detect potential malware “before it can execute.” *Id*. at 3:22-35, 8:30-33. The first phase is a “pre-execution process.” *Id*. at Fig. 4A, 3:28-32. Within the pre-execution process, an operating system kernel notifies a “pre-execution module” when “a new program begins to load into memory so that it can be validated before execution.” *Id*. at 3:45-47. After it is determined that a program is trying to load, a binary execution monitor suspends execution of the executable file “until the program can be validated.” *Id*. at 8:52-62.

41. The pre-execution process includes a validation module that validates the program by comparing the program to a “predetermined list of approved programs.” If the validation module determines that the program is valid (i.e., matches an entry in a predetermined list of approved programs), “little to no

additional security monitoring needs to be performed on the program while it is executing." *Id*. at 3:47-54. If the program is found within the predetermined list of approved programs, "the suspended state will be released." *Id*. at 9:4-6. A figure of the pre-execution process is shown below:



Id. at Fig. 4A.

42. The second phase of the protector system is concerned with "if the program is not validated." *Id*. at 3:54-57. After the pre-execution process terminates and the suspended state is lifted, "in step 605 the protector application 115 will consult the database 110 to determine if the user has been previously queried about the new non-validated executable file." *Id*. at 9:43-54. If the "user has previously approved the loading of this executable file," then execution of the non-validated file will proceed. *Id*. If the program is not found in the database, the program is allowed to execute while detect drivers "perform additional monitoring while the new program is executing" and "identify potential threats to the computing device before they are executed" at the operating system kernel of the computing device. *Id*. at 4:7-11, *see also* 6:64-67. "If the program is not validated, it can be monitored at the kernel level of the operating system during the second phase, while the program is executing." *Id*. at 3:30-35. Detection and monitoring modules "monitor the activities before they are able to cause harm to the computing device" and "identify suspicious activities triggered by the program." *Id*. An example of a non-validated execution process is shown below:

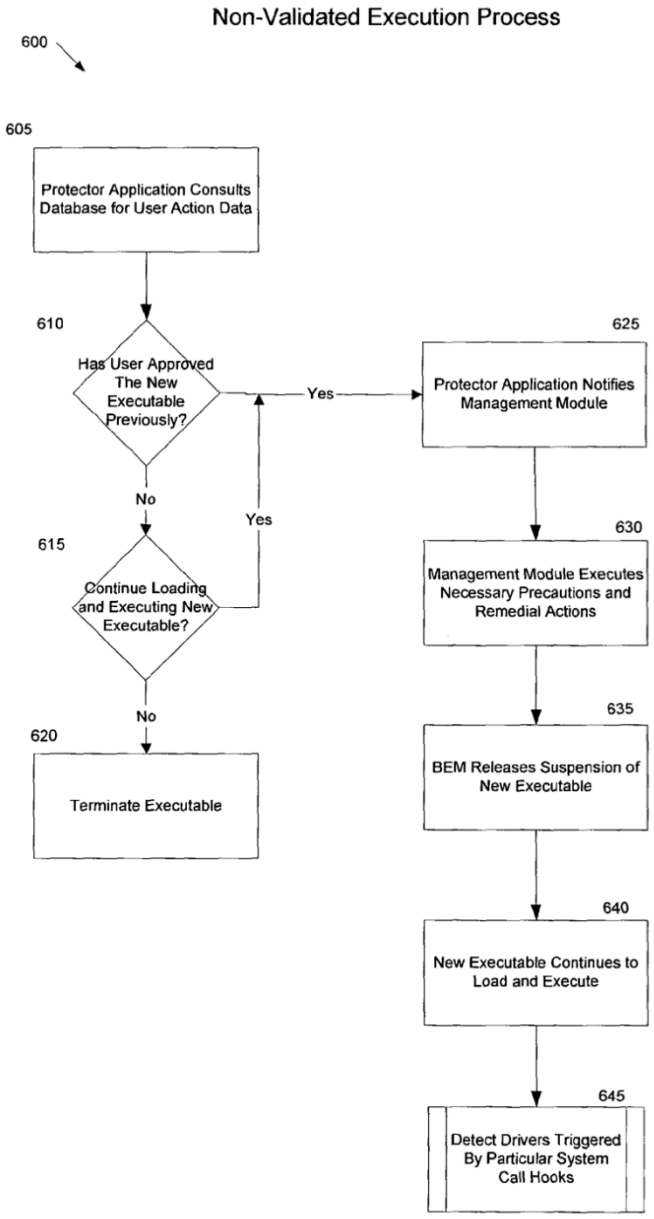


FIG. 6

Id. at Fig. 6.

B. Level of a Person Having Ordinary Skill in the Art

43. I have been asked to provide my opinion as to the level of skill of a person having ordinary skill in the art at the time of the '137 Patent, which I have

been instructed to assume is January 4, 2002. In determining the characteristics of a hypothetical person having ordinary skill in the art at the time of the claimed invention, I was told to consider several factors, including the type of problems encountered in the art, the solutions to those problems, the rapidity with which innovations are made in the field, the sophistication of the technology, and the education level of active workers in the field. I also placed myself back in the time frame of the claimed invention and considered the colleagues with whom I had worked at that time.

44. In my opinion, a person having ordinary skill in the art, as of January 4, 2002, would have been a person having 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. Such a person would have been capable of understanding the '137 Patent and the prior art references discussed below.

45. Based on my education, training, and professional experience in the field of the claimed invention, I am familiar with the level and abilities of a person of ordinary skill in the art at the time of the claimed invention. Additionally, I met at least these minimum qualifications to be a person having ordinary skill in the art as of the time of the claimed invention of the '137 Patent.

C. Background of the Prior Art

46. I was asked to briefly summarize the background of the prior art from the standpoint of a person having ordinary skill in the art prior to January 4, 2002.

1. Overview of Operating Systems

47. An operating system is "[t]he master set of programs that manage the computer." *Dictionary* at 411. "Among other things, an operating system controls input and output to and from the keyboard, screen, disks, and other peripheral devices; loads and begins the execution of other programs; manages the storage of data on disks; and provides scheduling and accounting services." *Id.* The kernel is the set of programs in the operating system "that implement the most primitive of that system's functions" and constitutes "the core of the operating system." *Id.* at 318.

48. Operating systems typically have multiple layers of privilege to protect the computer's resources. For example, in the 1990s, Intel released x86 processors having four "protection rings." *Intel System Programming Manual* at 4-8 ("The processor's segment-protection mechanism recognizes 4 privilege levels, numbered from 0 to 3. The greater numbers mean lesser privileges. Figure 4-2 shows how these levels of privilege can be interpreted as rings of protection."). "The processor uses privilege levels to prevent a program or task operating at a lesser privilege level from accessing a segment with a greater privilege, except under controlled situations." *Id.*

As shown in the figure below, "[t]he center (reserved for the most privileged code, data, and stacks) is used for the segments containing the critical software, usually the kernel of an operating system," while the "[o]uter rings are used for less critical software." *Id.*

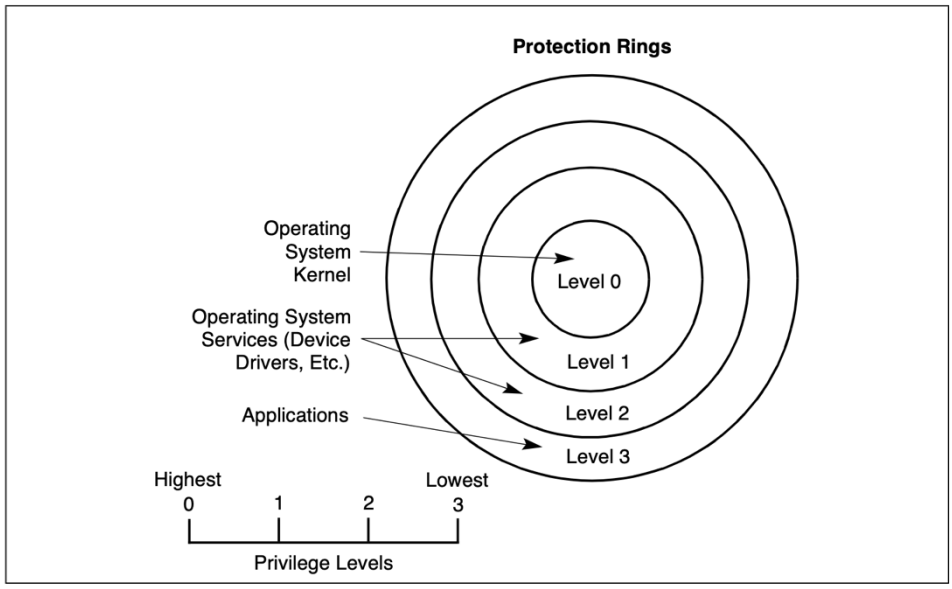
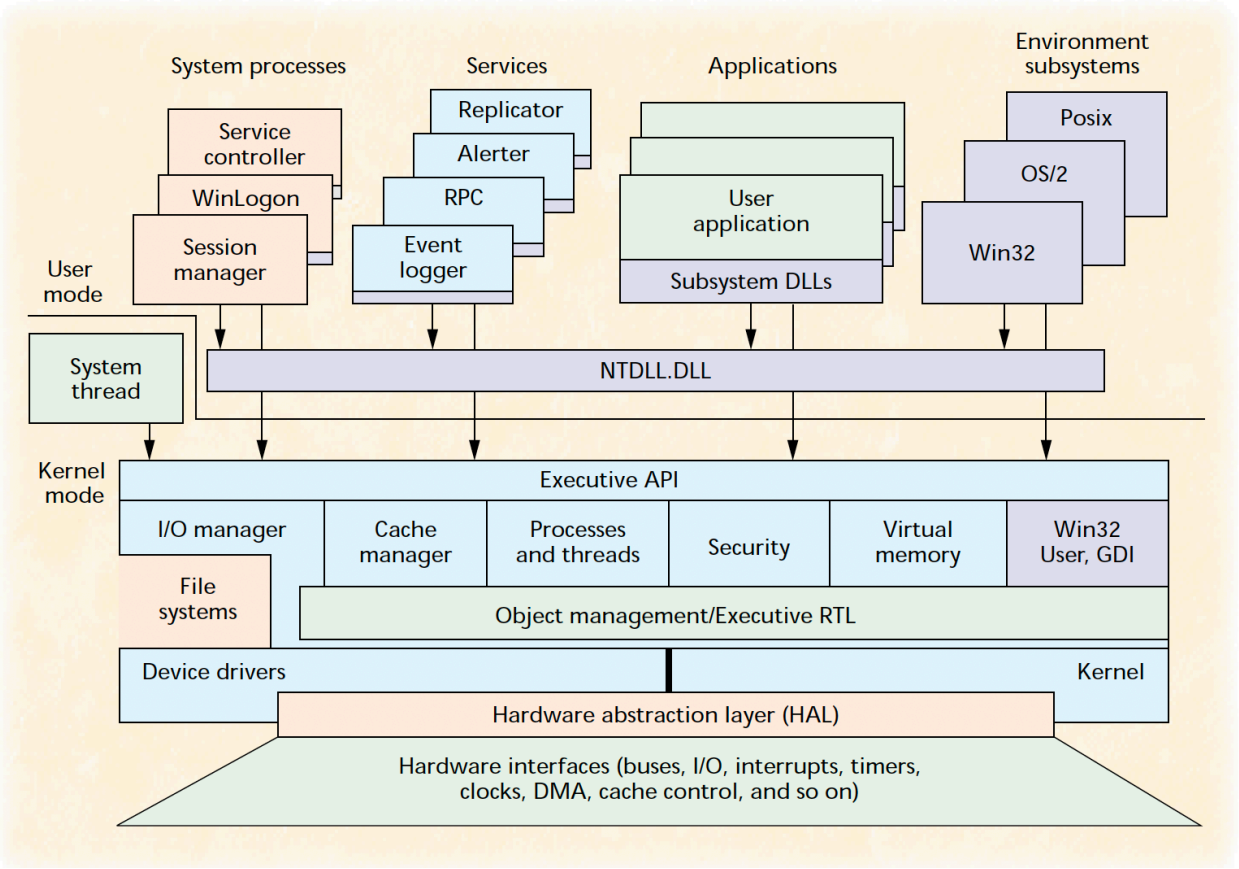


Figure 4-3. Protection Rings

Id.

49. Not all operating systems implemented all four protection rings. For example, the Microsoft Windows NT 5.0 operating system (aka Windows 2000), which was launched on February 17, 2000, only utilized ring 0 (i.e., kernel mode) and ring 3 (i.e., user mode):



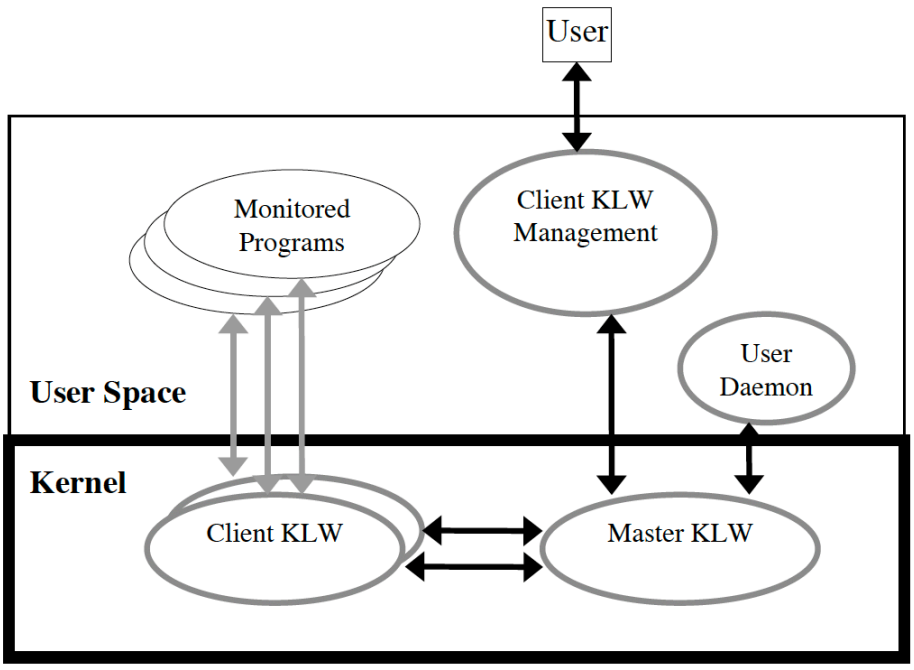
Solomon at 41; *see also, Neshevich* ("Microsoft has changed the name of its Windows NT 5.0 platform to Windows 2000, with the intent to make it Microsoft's mainstream operating system for both consumers and businesses in the new millennium."); *see also, Microsoft Windows 2000 Press Release* at 1 ("Microsoft plans general availability of Windows 2000 with a worldwide launch on Feb. 17, 2000.").

50. As shown in the figure above, user applications, services, and other user mode components accessed the kernel mode via the NTDLL.DLL. *Solomon* at 41. The NTDLL "is a special system support library" that contains "architecture-specific

instruction that causes a transition into kernel mode to invoke the actual kernel-mode system service[.]" *Id.* at 45.

51. Programmers could directly access the kernel by writing custom device drivers that would be loaded into the kernel. *Id.* at 44 ("Because installing a device driver is the only way to add user-written kernel-mode code to the system, some programmers have written device drivers simply as a way to access internal operating system functions or data structures that are not accessible from user mode.").

52. Like the Windows NT architecture, the Linux operating system's architecture similarly includes separate kernel and user spaces as illustrated in the figure below:



- Master KLW provides communications and control
- Client KLWs provide application specific monitoring
- Client KLW Management provides the user interface for configuration of client KLWs

Figure 1. KLW architecture

Mitchem at 3.

2. ***Computer Viruses and Malware in the Internet Era***

53. A virus is "[a]n intrusive program that infects computer files by inserting in those files copies of itself. The copies are usually executed when the file is loaded into memory, allowing the virus to infect still other files, and so on. Viruses often have damaging side effects[.]" *Microsoft Computer Dictionary* at 500.

54. "In the early days of computing and certainly before the advent of the Internet, viruses and all of their variations were usually spread physically, through

the loading of files from a floppy disk." *Wiggins* at 4. For example, in 1986, a virus called the "Brain" replaced the boot sector of a floppy disk with a copy of the virus. *Id.* at 2 ("1986 – Two brothers from Pakistan analyzed the boot sector of a floppy disk and developed a method of infecting it with a virus dubbed 'Brain'."). Thus, spreading viruses and malware was harder in the early days of computing since it required the user to physically load an infected floppy disk into their computer.

55. As the internet's popularity and availability grew in the late 1990s and early 2000s, so did the threat of malware. By the year 2001, e-mail had "become the most popular form of transmission." *Id.* at 4. For example, in 1999, the "Bubbleboy" virus became "the first worm that would activate when a user simply opened an e-mail message in Microsoft Outlook and previewed the message in Outlook Express." *Id.* at 3.

56. In the year 2000, "the first major distributed denial of service attacks shut down major websites such as Yahoo! and Amazon.com." *Id*. at 3. At the same time, the Loveletter worm "became the fastest-spreading worm; shutting down e-mail systems around the world." *Id*. A worm is a form of malware that self-replicates over a network, adding itself to a vulnerable system, and using the network to start scanning for other vulnerable systems. *See, e.g.*, *Microsoft Computer Dictionary* at 261 (defining "Internet worm" as "A string of self-replicating computer code that was distributed through the Internet in November 1988. In a single night, it

overloaded and shut down a large portion of the computers connected to the Internet at that time by replicating itself over and over on each computer it accessed, exploiting a bug in UNIX systems.); 512 (defining "worm" as "A program that propagates itself across computers, usually by creating copies of itself in each computer's memory.").

57. At this time, attacks on the kernel of an operating system, often in the form of trojan horses, became more common. *Iglio* at 1. ("On many hacker Web sites it is explained how to use any executable program as a *trojan horse* to infect a Windows system using NetBus or BackOrifice. Some software tools are even publicly available to simplify this task. Once the system has been infected, hackers can remotely read all keystrokes, read and upload files, etc. on the infected machine.") A trojan horse is "[a] destructive program disguised as a game, utility, or application. When run, a Trojan horse does something harmful to the computer system while appearing to do something useful." *Microsoft Computer Dictionary* at 478. "The earliest Trojan horse programs were bundled together in the form of "Root Kits"." *Manap* at 4.

58. Another type of malware that was seeing an increase of deployment was the rootkit. Rootkits "hide an attacker['s] presence and at the same time give the attacker ability to keep full control [of] the server or host continuously without being detected." *Manap* at 1. There are two categories of rootkits, including an

"Application rootkit – established at the application layer" and a "Kernel rootkit – establish more deep into kernel layer." *Id.* at 5. The application rootkit replaces "the good system application with a trojaned system file," which provides a "backdoor, hiding the attackers presence, and it also will not log any connection and activity done by the attacker." *Id*. A Kernel rootkit manipulates and exploits the kernel and is the hardest type of rootkit "to detect because it can bypass conventional system integrity checker at [the] application layer." *Id*. at 6.

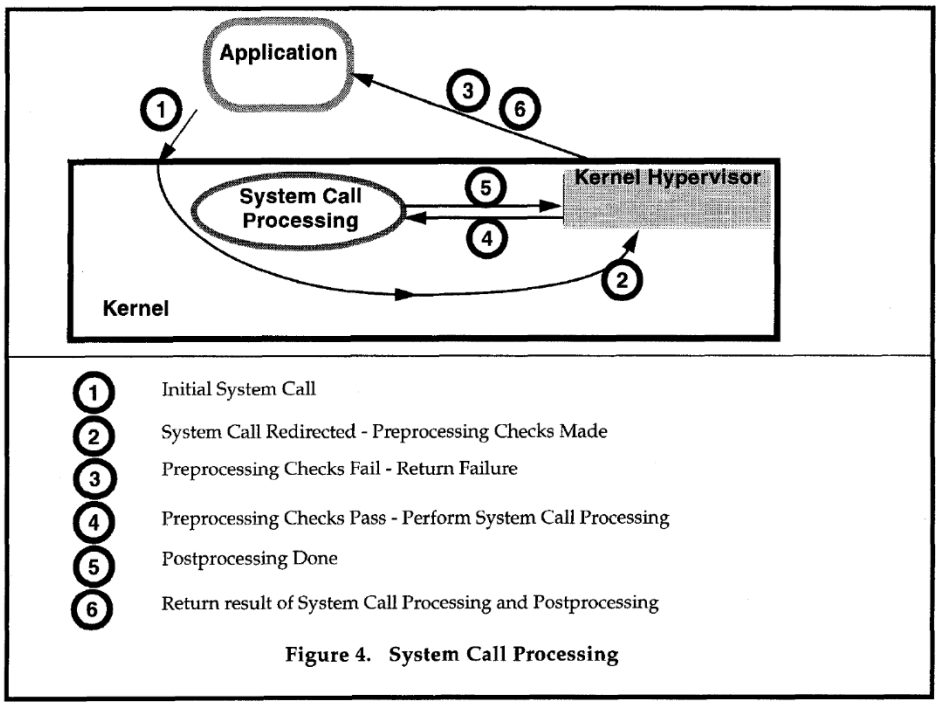
59. Unfortunately, restricted security policies were often not practical because of "the need of using the same personal computer to edit documents, to surf the Web, to exchange e-mail, to run multimedia applications, and to perform critical tasks." *Igilo* at 1. "For example, it is a common practice to use the same personal computer to run untrusted applications (such as applications downloaded from the network) and to buy products through sites offering e-commerce services." *Id.* However, this practice was "very dangerous" and gave hackers the opportunity to "break into systems and replace critical components to gain remote control of the system and to leave hidden backdoors for future access." *Id.*

3. ***Computer Monitoring Systems***

60. Monitoring systems were developed in response to the rapid increase of the malware threats described above. Specifically, kernel monitoring systems were developed to block malicious attacks from harming the computing device. For

example, Mitchem proposed using kernel hypervisors to "provide unbypassable security wrappers for application specific security requirements[.]" *Mitchem* at Abstract. Mitchem proposed using such kernel hypervisors for "a number of potential applications," including, for example, "protecting user systems from malicious active content downloaded via a Web browser." *Id.* "Kernel hypervisors are loadable kernel modules that intercept system calls to perform pre-call and post-call processing." *Id.* at 175. Kernel hypervisors are also "easy to install, requiring no modification to the kernel or to the COTS [Commercial Off The Shelf] applications that they are monitoring." *Id.* Moreover, "[k]ernel hypervisors can be used on any operating system that supports kernel loadable modules," including "Linux, Solaris, and other modern Unix systems, as well as Windows NT." *Id.* at 177.

61. The monitoring of the system calls is "performed by redirecting the links in the kernel system call table to point to system call wrappers[.]" *Id*. at 178. "The wrapper code is invoked when the system call is made and performs the processing illustrated in Figure 4." *Id.*



Id. at Figure 4.

62. A similar kernel monitoring system was proposed by Bernaschi. Per Bernaschi, "immediate detection of security rules violations can be achieved by monitoring the system calls made by processes, and blocking any malicious invocations of system calls from being completed." *Bernaschi* at 174. To accomplish this, Bernaschi proposes "to 'instrument' the system calls so that the system call itself once invoked checks whether the invoking process and the value of the arguments comply with the rules kept in a small access control database within the kernel." *Id.* As a proof of concept, Bernaschi "implemented a prototype for monitoring system calls which blocks buffer overflow attacks before they can complete." *Id.* However, Bernaschi's "OS reference monitor is intended to protect

against any technique that allows an attacker to *hijack the control* of a privileged process." *Id.* (emphasis in original).

63. Thus, it was well understood that kernel-level system call monitoring was an effective technique for detecting and blocking many different types of malware attacks well prior to the '137 Patent.

D. Claim Construction

64. I have been asked to apply the following claim constructions in my analysis:

Claim Limitation	Construction
"if the new program is validated, permitting the new program to continue loading and to execute in connection with the computing device" (Claim 6)	"if the new program is validated, then it is not monitored while it loads and executes in connection with the computing device"
"if the new program is valid, permitting the new program to execute on the computing device" (Claim 25)	"if the new program is valid, then it is not monitored while it executes on the computing device"
"pre-execution monitor" (Claim 1)	"pre-execution module"
"an execution module coupled to the detection module and operable for monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module" (Claim 1)	Means-plus-function applies. Function: "monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module" Structure: "if the program was not validated, then monitor the non-validated program in response to triggers while the program is executing"

65. For all other claim limitations, I have applied the ordinary and customary meaning of the claim terms as understood by a person having ordinary skill in the art.

E. Summary of the Prior Art References

1. Dan

66. Dan recognizes that the "[s]haring of unknown programs creates an atmosphere of untrust and hence exacerbates the need to secure information and physical resources from any *alien* code." *Dan* at Abstract (emphasis in original). According to Dan, "[t]he rapid growth in connectivity and sharing of information via the World-Wide Web has dramatically increased the vulnerability of client machines to *alien* codes." *Id.* at 1 (emphasis in original). "Once the alien code is installed on the client machine, it can run with all the privileges of the invoking user." *Id.* "The alien code can silently obtain unauthorized access to all the system resources (i.e., *logical resources* such as files, network ports) that can be accessed by the invoking user." *Id.* (emphasis in original)

67. To solve this problem, Dan proposes "a generic framework that allows the operating system to authenticate and verify installed applications, dynamically control access to both logical and physical resources, and allow privileges associated with an application to be transferred to any other application and/or system." *Id.* at 4. Dan's "system consists of one or more code production systems, certification

agencies, servers and clients." *Id.* at 5. "A code production system communicates with a certification agency, which is a trusted third party," and "[t]he certification agency issues a certificate for the code identifying its source and a certificate for the RCL of that code." *Id.* A Resource Capability List (RCL) is "the set of permissions and resource access privileges required by the code." *Id.* at 4. "The code and its RCL are stored on a server along with their certificates." *Id.* at 5. As shown in Figure 1 below, the code, the code's certificate, the RCL, and the RCL's certificate are downloaded to a client system:

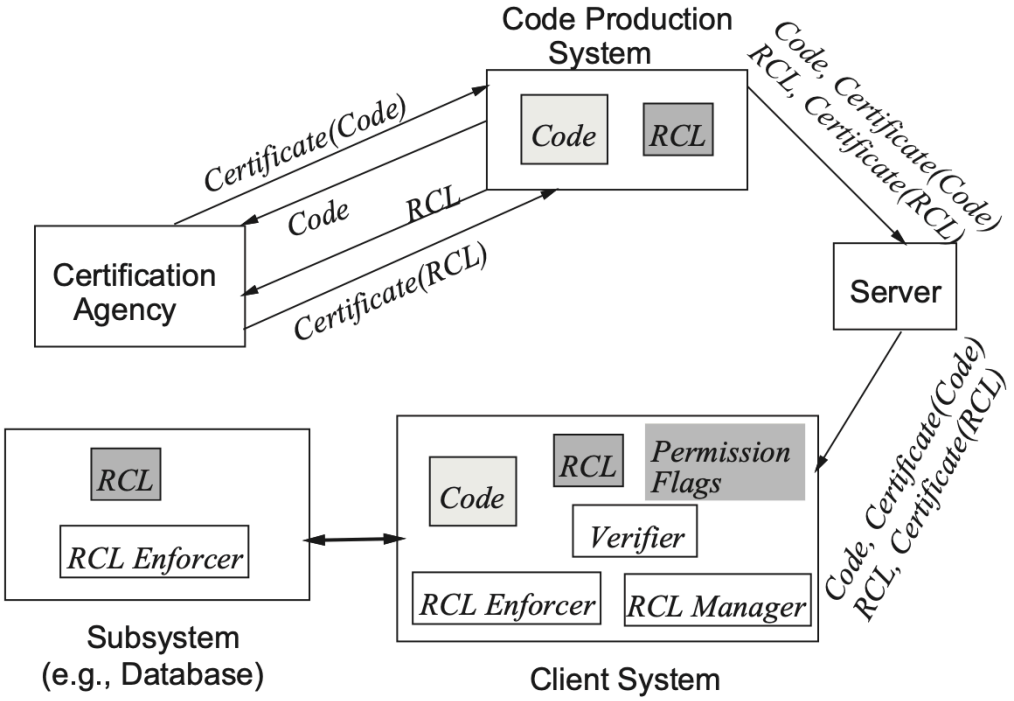


Figure 1: A. Block diagram of the proposed system.

Id. at Figure 1.

68. The client system includes a sandbox environment having three primary components including the verifier, RCL manager, and RCL enforcer. *Id.*; see *also, id.* at 5 ("[T]he RCL may be the default RCL associated with the sandbox environment in the client system."). "When a client receives the code and its RCL, it invokes the verifier to verify that the code and RCL are valid (not tampered with)." *Id.* at 5. For example, the verifier module on the client system may first check to see if the Certification Agency (CA)'s "signature on the certificate is valid (using the CA's known public key)" and "then compute the cryptographic hash of the code/RCL and verif[y] that it matches that in the certificate." *Id.* at 7.

69. "Once the code is verified, the RCL manager is responsible for determining which subset of parameters within the RCL are to be allowed." *Id.* at 5-6. The RCL manager "then stores the code and its modified RCL in a secure location." *Id.* at 6.

70. "The RCL enforcer is invoked when the code is executed." *Id.* "The RCL enforcer module at the client ensures that the permissions and resource consumption limits specified in the RCL are not violated." *Id.* at 9. The RCL enforcer "monitors two types of resources: physical and logical." *Id.* For example, the "RCL enforcer protects physical resources such as memory and CPU usage, disk storage, and disk bandwidth utilization." *Id.*

71. The RCL enforcer also monitors logical resources such as "file, directories, communication ports and various other system service calls." *Id.* at 12. To accomplish this, "[t]he RCL enforcer monitors all system calls, i.e., access to requests to logical resources in the system." *Id.* "This monitoring requires that the RCL enforcer be invoked for each system call made by an application." *Id.* "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT [Runtime Logical Resources Table] data structures and either allows or disallows the call from continuing." *Id.*

72. Dan describes "three different approaches to monitoring system calls," including "Binary code modification," "Modification of service libraries," and "System call modification." *Id.* at 13-14.

73. The binary code modification approach requires the downloaded application code to be "preprocessed and modified to insert hooks in the binary code for calling RCL enforcer (i.e., `CheckAccessRequest()` function. See, Figure 7 and 8)." *Id.* at 14. However, this approach requires "the user to preprocess all untrusted applications," which "may incur a lot of overhead." *Id.* "Also, this is not a full [sic] proof solution, since a program may use encryption to hide system calls in its code." *Id.*

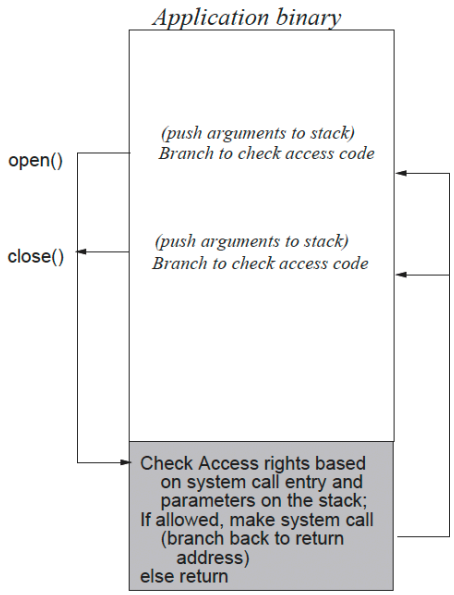


Figure 8: Preprocessor for modifying an application's binary code.

Id. at Figure 8.

74. The service library modification approach requires providing "in each machine a duplicate set of libraries for various services that may make system calls (e.g., file system, language support, etc.)." *Id.* "The new set of libraries calls the `CheckAccessRequest()` routine to verify the parameters before making any system call." *Id.* However, "the system cannot enforce security for applications that have system call libraries statically linked into the application." *Id.*

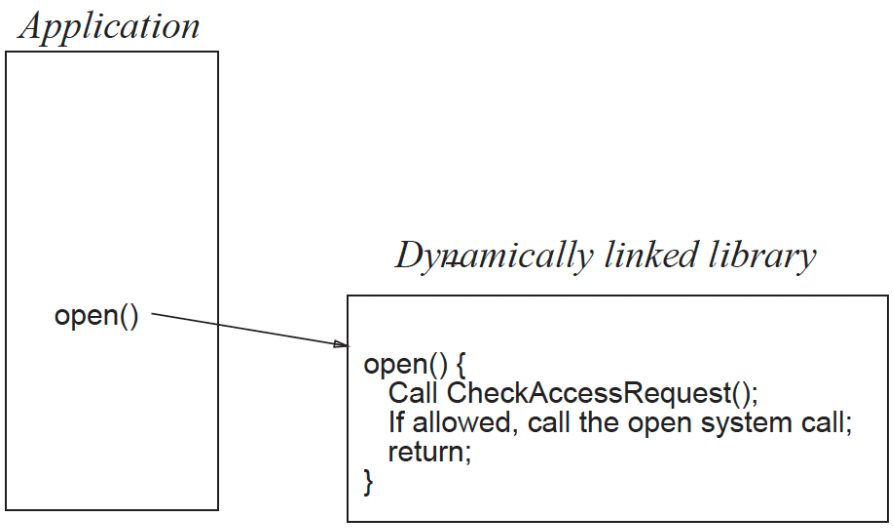


Figure 9: The modified system call library approach.

Id. at Figure 9.

75. In the system call modification approach, "the operating system's call interface is modified to call the `CheckAccessRequest()` upon entry to verify the parameters of the system call." *Id.* at 14. "Thus, regardless of how the applications are structured, the system call entry point in the operating systems will ensure consistent security checks for all applications that are executed on the machine." While this approach requires modifying the operating system, Dan "selected this approach to manage logical resources in the system primarily because of the consistency it offers for enforcing security." *Id.* at 14-15.

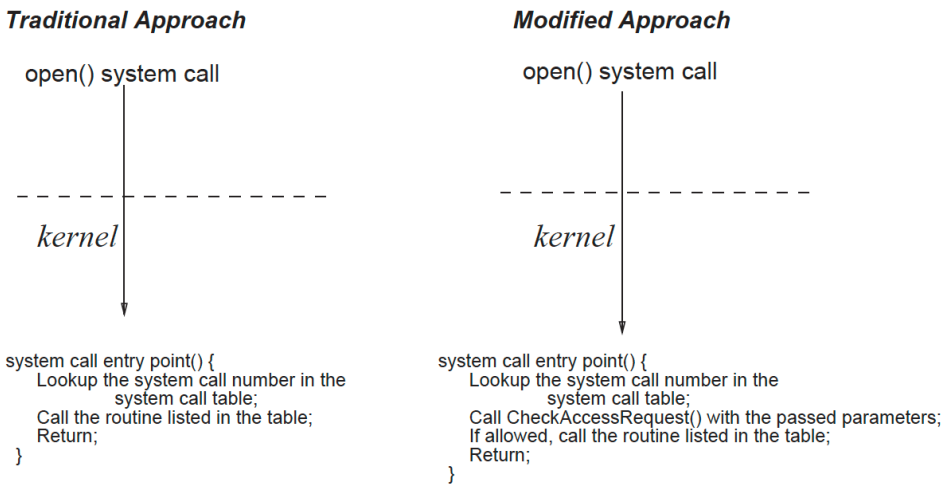


Figure 10: The modified operating system call interface approach.

Id. at Figure 10.

76. I note that in more recent times, the term "sandbox" can sometimes refer to a virtual environment that simulates a computer's operating system and hardware. *See, e.g., STL Tech* (describing full-system emulation, integration with operating systems, full virtualization, browser-based sandboxing at 1). However, it is clear from Dan's disclosure that the client sandbox is not operating in a virtual environment because Dan's RCL enforcer directly monitors a program's access of the computer's real physical resources, including consumption of disk, CPU, and memory. *Dan* at 4, 9-11. Dan's RCL enforcer also monitors system calls made to the computer's operating system – not an emulated version of the operating system. *Id.* at 14-15.

77. As I discussed above, I have been informed that for a prior art reference to be proper for use in an obviousness analysis, the reference must be "analogous

art" to the claimed invention. I have been informed that a prior art reference is analogous to the claimed invention if, for example, the reference is from the same field of endeavor as the claimed invention, and/or directed at solving similar problems.

78. Both Dan and the '137 Patent are related to systems and methods for analyzing and resolving potential security threats to a computing device. *Compare* '*137 Patent* at Abstract ("[m]anaging and controlling the execution of software programs with a computing device to protect the computing device from malicious activities"), 3:5-14 ("there is a need in the art for a security system which will provide early detection of security threats"), 6:21-24 ("the command line interface 120 is to configure various security settings, such as how to respond to certain threat"), 7:15-21 ("the behavior monitor 128 can take direct action to address a security threat or instruct the protector application 115 to query the user for instructions on how to handle the threat"), 8:20-27 ("the configuration settings can include predetermined responses to particular threats and decision rules as to when the user should be queried about a security threat") with *Dan*, 21 ("[h]ence, the alien code can silently obtain unauthorized accesses to all the system resources (e.g. files, network ports) that can be accessed by the invoking user. The alien code may propagate itself, attempt illegal break-ins or maliciously alter files"), 5 ("[w]hen a client receives the code and its RCL, it invokes the verifier to verify that the code

and RCL are valid (not tampered with)"), 1 ("it may still engage in a *denial of service* attack by monopolizing *physical resources* such as disk, physical memory, CPU, network bandwidth"), 2 ("[a] similar security problem exists with importing data for word processors and spreadsheets, where the imported data could be infected with *active content macro viruses*, and these viruses in turn, could try to access system resources without the knowledge of the user"), 12 ("[t]he RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system." "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT data structures and either allows or disallows the call from continuing"), 14-15 ("[t]hus, regardless of how the applications are structured, the system call entry point in the operating systems will ensure consistent security checks for all applications that are executed on the machine" "The common entry point to the Kernel for all system calls provides an effective mechanism for enforcing access control for logical resources"). A person having ordinary skill in the art would have recognized Dan is in the same field of endeavor as the '137 Patent.

79. Dan is also directed at solving similar problems to those of the '137 Patent, such as detecting potential security threats to a computing device. *Compare* '137 Patent at 3:5-14 ("there is a need in the art for a security system which will provide early detection of security threats"), 3:25-35("[i]mplementing a two-phased

approach at the kernel level of the computing device's operating system, the present invention provides fast and efficient security that minimizes interruptions for the user"), 4:55-60 ("[t]he two-step process provides an efficient and effective means for protecting a computing device or network while minimizing disruptions for the user"), 10:49-54 ("the present invention enables and supports security from malicious software programs for a computing device") with *Dan* at 2 ("we propose a solution to the above security problem. The solution combines certification of alien code together with explicit granting of privileges for execution"), 5-6 ("[w]hen a client receives the code and its RCL, it invokes the verifier to verify that the code and RCL are valid (not tampered with)"), 12 ("[t]he RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system." "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT data structures and either allows or disallows the call from continuing"), 14-15 ("[t]hus, regardless of how the applications are structured, the system call entry point in the operating systems will ensure consistent security checks for all applications that are executed on the machine" "The common entry point to the Kernel for all system calls provides an effective mechanism for enforcing access control for logical resources"), 21 ("[s]haring of unknown programs creates an atmosphere of untrust and hence, exacerbates the need to secure information and physical resources from *alien* codes"), Figure 1. For these reasons, a person having

ordinary skill in the art would have recognized Dan is directed to solving similar problems to those of the '137 Patent.

2. **Lambert**

80. Lambert recognizes that "[c]omputer users, and particularly business users, typically rely on a set of programs that are known and trusted." *Lambert* at [0003]. "For example, in enterprise computing environments, system administrators and helpdesk personnel are trained in supporting a core set of programs used in operating the business." *Id.* "However, in most enterprise environments, particularly in larger environments in which some flexibility is important with respect to what various employees need to do with computers to perform their jobs, users are able to run unknown and/or untrusted code, that is, programs outside of the supported set." *Id.* These untrusted programs are problematic because "some of the untrusted code may be malicious code (e.g., a computer virus possibly in the form of a Trojan horse) that intentionally misrepresents itself to trick users into running it, typically inflicting damage, corrupting data or otherwise causing mischief to the computers on which the malicious code runs." *Id.* at [0004]. "The problem or running unknown code is compounded by recent developments in computing, wherein many types of content that formerly consisted of only passive data (e.g., documents and web pages) have potentially become executable code, due to the scripts and/or macros they may contain." *Id.* at [0005].

81. To solve these problems, Lambert "provides an improved computer security system and method wherein software that possibly includes executable code is identified and classified, and then a rule corresponding to the classification (or if none correspond, a default rule) automatically and transparently determines the software's ability to execute." *Id.* at [0008]. First, "[a]n administrator identifies/classifies the software and sets the rules, including a default rule that applies to any software file not classified under a specific rule." *Id.* at [0052]. "[O]ne way that a software file can be identified (and thus matched to a corresponding rule) is by a hash of its contents, which is essentially a fingerprint that uniquely identifies a file regardless of whether the file is moved or renamed." *Id.* at [0055]. "Via a user interface or the like operably connected to a suitable hash mechanism, an administrator can select any software file, obtain a hash value from its contents, and enter and associate a rule (that specifies a security level) with that hash value." *Id.*

82. The rule associated with the software file dictates whether the software file is allowed to execute in an unrestricted manner, allowed execute in a restricted manner, or not allowed to execute at all. "In the case of known bad code, the administrator may set the rule to specify that any file with that hash will not be allowed to execute at all (i.e., 'disallowed' or 'revoked' status)." *Id.* at [0056]. "Alternatively, in the case of known good code that is allowed to execute, a hash

rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent)." *Id.*

83. "Multiple files can have their corresponding hash values and rules maintained in a policy object, or otherwise made available to the system, whereby for any given instance of any file, the set of maintained hash values can be queried to determine whether a rule exists for it." *Id.* at [0055]. "In other words, before opening a file for execution, a hash value is determined from that file's contents, and the set of hash value information (e.g., in the policy object) searched to determine if a rule exists for that hash value. If so, that rule provides the security level for the software." *Id.*

84. The process of identifying and classifying a software file is depicted in Figure 5A. Specifically, "When the process 508 wants to load and possibly execute the software file 510, it identifies itself and provides information 514 identifying the software file to an appropriate function 516 (of a set) that can open, load and/or execute the file 510." *Id.* at [0066]. "For example, this can be accomplished by placing an application programming interface (API) call to the operating system, wherein the file information 514, typically in the form a path and filename is passed, as a parameter." *Id.*

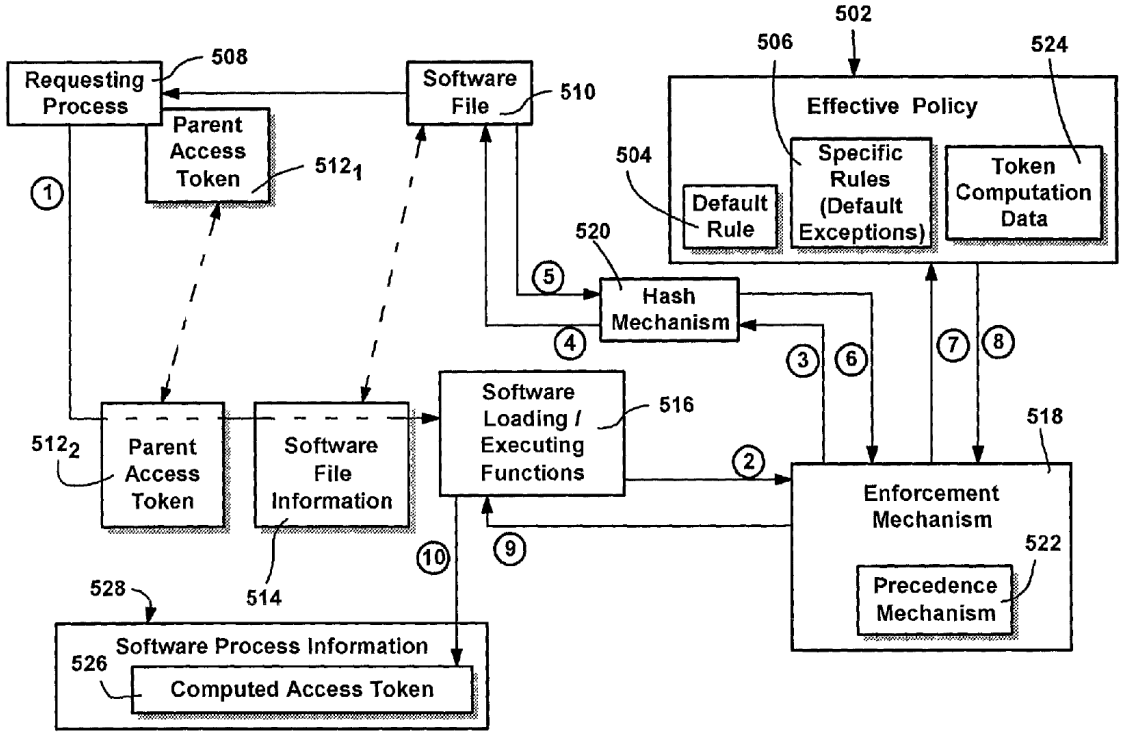


FIG. 5A

Id. at Fig. 5A.

85. "[I]nstead of simply loading the file and executing any executable code therein with the parent token's rights and privileges, each of the functions 516 is arranged to first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG. 5A)." *Id.* at [0067]. "[B]ased on this information the enforcement mechanism consults the effective policy 502 to determine which rule applies for the file 510, and from the rule determines whether to open/execute the file, and if so, the extent of any restricted execution context for the file 510." *Id.*

86. If the software file is associated with a rule that restricts its execution, "the enforcement mechanism 518 can compute and associate a restricted token with the software file 510, whereby if creation of a process is requested for the software file, (it includes executable content), any process of the software file is restricted in its access rights and privileges according to the restricted token." *Id*. at [0070]. "[T]he enforcement mechanism 518 completes the call and returns information back to the software function 516 that called it, the returned information essentially including an instruction on whether to open the file, and if so, the token to associate with the file." *Id.* at [0071]. "If the computed token 526 is a restricted token, the process of the software file can now execute, but only in association with the restricted token, thus providing the rule-determined secure execution environment." *Id.*

87. Thus, Lambert describes a pre-execution phase where the system determines whether a software file may execute and, if so, whether it may execute in an unrestricted or restricted manner.

88. Lambert also discloses an execution phase where a restricted program is monitored to determine whether it may access a resource based on its restricted token. "[I]n FIG. 3, a process 300 may be associated with the restricted token 210 rather than the user's normal token 200, as determined by the operating system, such as when the process is untrusted." *Id.* at [0042]. "When the process 300 desires

access to an object 302, the process 300 specifies the type of access it desires (e.g., obtain read/write access to a file object), and the operating system (e.g., kernel) level provides its associated token to an object manager 304." *Id.* "The object 302 has a security descriptor 310 associated therewith, and the object manager 304 provides the security descriptor 306 and the restricted token 210 to a security mechanism 308." *Id.* at [0043]. "The security mechanism 308 compares the security IDs in the restricted token 210 along with the type of action or actions requested by the process 300 against the entries in the DACL [Discretionary Access Control List] 312." *Id.* "If a match is found with an allowed user or group, and the type of access desired is allowable for the user or group, a handle to the object 302 is returned to the process 300, otherwise access is denied." *Id.*

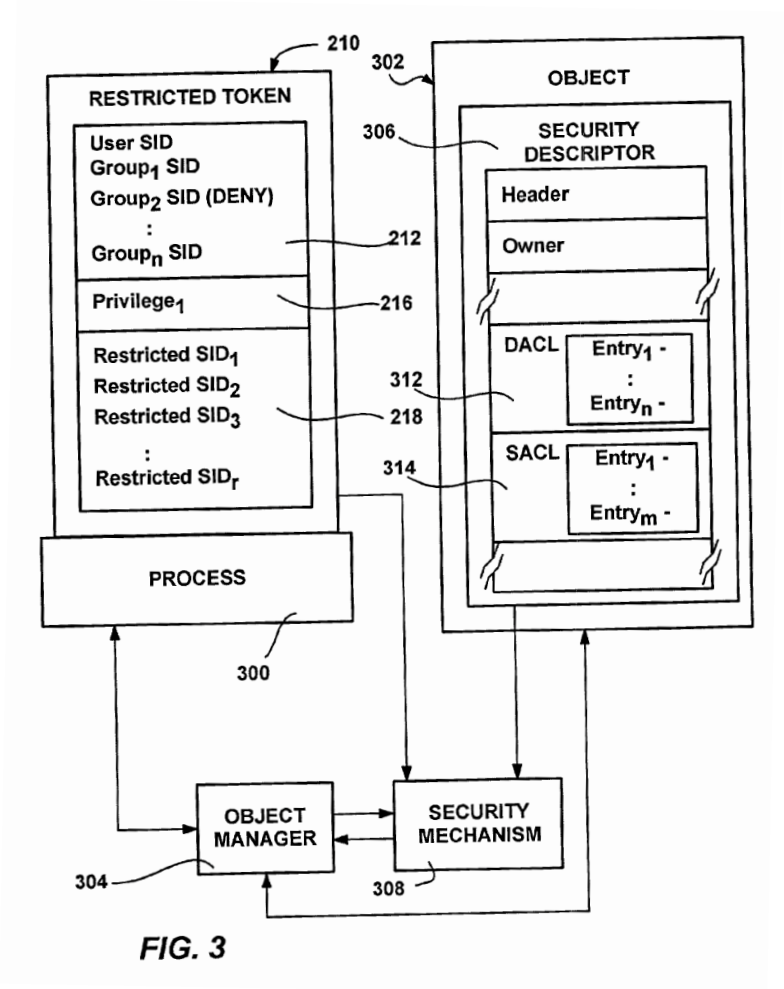


FIG. 3

Id. at Fig. 3.

89. As I discussed above, I have been informed that for a prior art reference to be proper for use in an obviousness analysis, the reference must be "analogous art" to the claimed invention. I have been informed that a prior art reference is analogous to the claimed invention if, for example, the reference is from the same field of endeavor as the claimed invention, and/or directed at solving similar problems.

90. Lambert and the '137 Patent are related to systems and methods for analyzing and resolving potential security threats to a computing device. *Compare* '137 Patent at Abstract ("[m]anaging and controlling the execution of software programs with a computing device to protect the computing device from malicious activities"), 3:5-14 ("there is a need in the art for a security system which will provide early detection of security threats"), 6:21-24 ("the command line interface 120 is to configure various security settings, such as how to respond to certain threat"), 7:15-21 ("the behavior monitor 128 can take direct action to address a security threat or instruct the protector application 115 to query the user for instructions on how to handle the threat"), 8:20-27 ("the configuration settings can include predetermined responses to particular threats and decision rules as to when the user should be queried about a security threat") with *Lambert* at Abstract ("[a] system and method that automatically, transparently and securely controls software execution by identifying and classifying software, and locating a rule and associated security level for executing executable software"), [0007] ("in contemporary computer systems, running unknown code cannot be prevented, and indeed is necessary to an extent in many enterprises"), [0009]("[t]he present invention provides various ways to identify/classify software, along with a highly flexible policy-based infrastructure to enforce decisions about whether software can run, and if so, how much the software will be restricted when it runs"), [0013]-[0014]

("[t]ypical security levels for execution (arbitrarily named herein) may include normal (unrestricted), constrained (restricted) or disallowed (revoked), although other levels such as end-user and untrusted are possible, and customized levels can be defined" "prior to loading a file, the various functions (such as of an operating system) that enable software to be loaded or otherwise executed are arranged to automatically and transparently communicate with an enforcement mechanism that determines the rule to apply for each file, based on the identification/classification information of the file"), [0041] ("a restricted token is derived from a parent token by removing at least one privilege therefrom relative to its parent token"), [0056] ("in the case of known good code that is allowed to execute, a hash rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent)" "one way that a software file can be identified (and thus matched to a corresponding rule) is by a hash of its contents"), Fig. 5A. For the reasons above, a person having ordinary skill in the art would have recognized Lambert is in the same field of endeavor as the claimed invention of the '137 Patent.

91. Lambert is also directed at solving similar problems to those of the '137 Patent, such as detecting potential security threats to a computing device. *Compare* '137 Patent at 3:5-14 ("there is a need in the art for a security system which will provide early detection of security threats"), 3:25-35("[i]mplementing a two-phased approach at the kernel level of the computing device's operating system, the present

invention provides fast and efficient security that minimizes interruptions for the user"), 4:55-60 ("[t]he two-step process provides an efficient and effective means for protecting a computing device or network while minimizing disruptions for the user"), 10:49-54 ("the present invention enables and supports security from malicious software programs for a computing device") with *Lambert* at [0004]-[0005] ("[t]he problem or running unknown code is compounded by recent developments in computing, wherein many types of content that formerly consisted of only passive data (e.g., documents and web pages) have potentially become executable code, due to the scripts and/or macros they may contain" "One problem with this approach is that it is reactive rather than proactive, in that a virus first needs to be individually identified and characterized by virus detection experts, after which it can have its appropriate signature information distributed to various computers to allow the automated detection thereof"), [0008] ("[t]he present invention provides an improved computer security system and method wherein software that possibly includes executable code is identified and classified, and then a rule corresponding to the classification (or if none correspond, a default rule) automatically and transparently determines the software's ability to execute"), [0011] ("[a] software restriction policy comprising a set of rules is provided to correlate the classification information for any software to a security level that may restrict or prevent the software from running. A policy can be per machine and per user, and can be

automatically applied to network users via group policy technology."), [0014] ("[t]he computation of the token or failing of the open may be automatic and transparent to the user, whereby security is implemented without requiring user forethought or action"). For the reasons above, a person having ordinary skill in the art would have recognized Lambert is directed at solving similar problems as the claimed invention of the '137 Patent.

3. **Schmid**

92. Schmid is generally directed to an execution management utility to "prevent software from executing without the prior approval of system administrator." *Schmid* at Abstract. Schmid describes, "most well-behaved Windows applications make system calls through the Win32 API, it is also possible to bypass this interface by making calls directly to kernel services." *Id.* at 5:29-32. On the other hand, a malicious program "may attempt to bypass a wrapper implemented between Windows and the Win32 subsystem by making direct calls to operating system objects, rather than through the Win32 API." *Id.* at 5:33-37.

93. To solve this problem, Schmid describes a "kernel-level wrapping approach to executable control" where a device driver is "installed in an I/O subsystem, and may be used to intercept system service requests." *Id.* at 5:38-48. The device driver is "dynamically loaded into a kernel, or may be loaded as part of a boot sequence" and "modifies an entry in a table consulted by a dispatcher to

handle an interrupt instruction." *Id.* at 6:6-11. "In the preferred Windows NT embodiment, the modified entry may cause a dispatcher to call an inserted function rather than ZwCreateProcess, the NT function responsible for creating new user- and kernel-mode processes." *Id.* at 6:11-15. Schmid describes "[a] kernel entity, known as a dispatcher, initially responds to an interrupt instruction, determines the nature of an interrupt, and calls a function to handle the request." *Id.* at 5:62-64. "When a substitute function is called by a dispatcher, the function determines whether to allow or block process creation." *Id.* at 6:22-24.

94. As I discussed above, I have been informed that for a prior art reference to be proper for use in an obviousness analysis, the reference must be "analogous art" to the claimed invention. I have been informed that a prior art reference is analogous to the claimed invention if, for example, the reference is from the same field of endeavor as the claimed invention, and/or directed at solving similar problems.

95. Schmid and the '137 Patent are related to systems and methods for analyzing and resolving potential security threats to a computing device. *Compare* Schmid at Abstract ("execution management utility designed to prevent software from executing without the prior approval of system administrative or other security staff"), 2:20-23 ("The present invention improves upon prior malicious software protection applications by not only resolving the problems discussed above, but also

by addressing emerging dangers and unknown attacks"), 2:31-33 ("For example, the present invention can assist corporations by enforcing policies regarding unauthorized, unlicensed, or pirated software"), 3:21-22 ("Using hashes for identification has the additional benefit of preventing modified applications from executing"), 2:50-54 ("improves upon existing executable control software, and provides administrators with a valuable defense against the introduction of hostile or unknown code"), Fig. 1 *with* '137 Patent at Abstract ("[m]anaging and controlling the execution of software programs with a computing device to protect the computing device from malicious activities"), 3:5-14 ("there is a need in the art for a security system which will provide early detection of security threats"), 6:21-24 ("the command line interface 120 is to configure various security settings, such as how to respond to certain threat"), 7:15-21 ("the behavior monitor 128 can take direct action to address a security threat or instruct the protector application 115 to query the user for instructions on how to handle the threat"), 8:20-27 ("the configuration settings can include predetermined responses to particular threats and decision rules as to when the user should be queried about a security threat"). For the reasons above, a person having ordinary skill in the art would have recognized Schmid is in the same field of endeavor as the claimed invention of the '137 Patent.

96. Schmid is also directed at solving similar problems to those of the '137 Patent, such as detecting potential security threats to a computing device. *Compare*

Schmid at Abstract (discussing preventing software from executing without prior approval), 2:20-23 (discussing addressing emerging dangers and unknown attacks), 2:26-28 (discussing a kernel intercepting process creation requests), 2:50-54 (discussing providing administrators with defense against hostile or unknown code), *with* '137 Patent at 3:5-14 ("there is a need in the art for a security system which will provide early detection of security threats"), 3:25-35("[i]mplementing a two-phased approach at the kernel level of the computing device's operating system, the present invention provides fast and efficient security that minimizes interruptions for the user"), 4:55-60 ("[t]he two-step process provides an efficient and effective means for protecting a computing device or network while minimizing disruptions for the user"), 10:49-54 ("the present invention enables and supports security from malicious software programs for a computing device.") For the reasons above, a person having ordinary skill in the art would have recognized Schmid is directed at solving similar problems as the claimed invention of the '137 Patent.

F. Ground 1: The Combination of Dan and Lambert Renders Claims 6, 8, 9, 12, and 25 Obvious

1. Claim 6

(a) *6[Pre]. – A computer implemented method for implementing security for a computing device comprising the steps of:*

97. Dan discloses a client system that includes a sandbox environment having three primary components, including the verifier, RCL manager, and RCL enforcer:

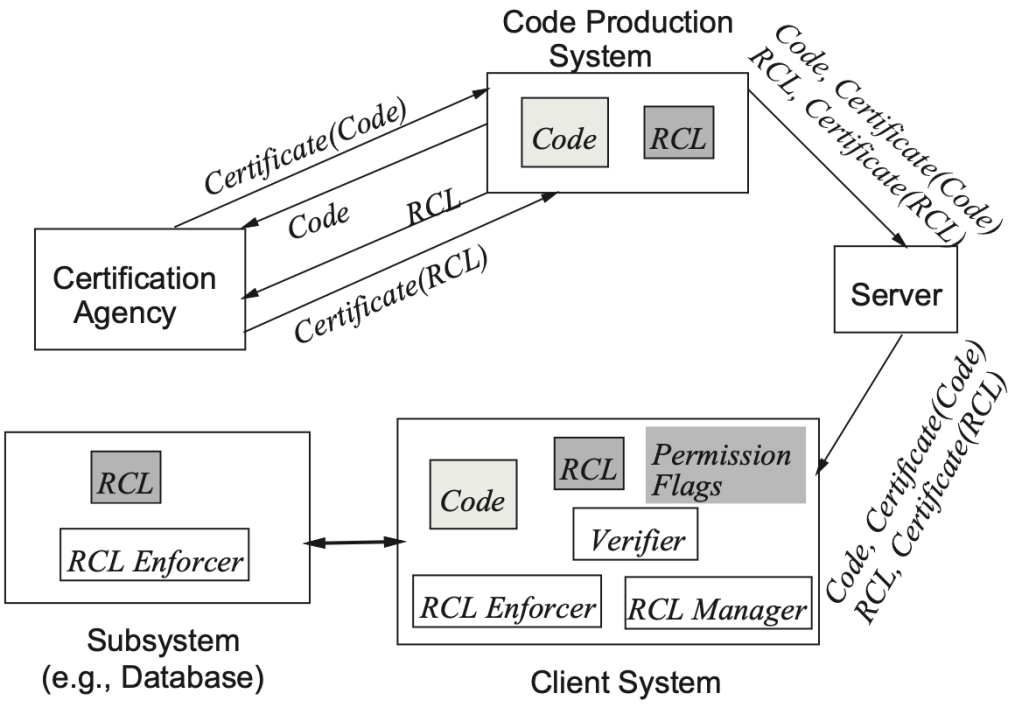


Figure 1: A. Block diagram of the proposed system.

Dan at Figure 1; see *also, id.* at 5 ("[T]he RCL may be the default RCL associated with the sandbox environment in the client system."); Section III.E.1.

98. Dan's client system is clearly *a computing device*.[1] For example, Dan teaches that the client system includes an "operating system kernel," logical resources, such as "files [and] network ports," and physical resources, such as "disk, physical memory, [and] CPU." *Dan* at 5, 7; *see also, id.* at 1 (describing a client

[1] I have underlined and *italicized* claim language for clarity.

machine as having "system resources (i.e., logical resources such as files, network ports)" and "physical resources such as disk, physical memory, CPU, network bandwidth."). As I discussed above, the operating system is the master set of programs that manage a computer, and the kernel is the set of programs in the operating system that implement the most primitive of that system's functions. *See*, Section III.C.1. In fact, an operating system kernel makes no sense without a computing device, including a processor executing the operating system kernel, which would be stored in an associated storage.

99. Likewise, the disk, memory, and CPU are all well-known components of computers. *See, e.g., Dictionary* at 79 (defining "central processing unit (CPU)" as "[t]he major component of a computer system with the circuitry to control the interpretation and execution of instructions."), 179 (defining "disk" as "[a] magnetic device for storing information and programs accessible by a computer."), 364 (defining "memory" as "[s]torage facilities of the computer, capable of storing vast amounts of data."). As such, a person having ordinary skill in the art would have understood that Dan's client system is *a computing device*.

100. The sandbox components on Dan's client system perform *a computer implemented method for implementing security for a computing device*. As I discussed above in my summary of Dan, the client system "invokes the verifier to verify that the code and RCL are valid (not tampered with)" when the client receives

the code and its RCL. *See*, Section III.E.1; *Dan* at 5. "Once the code is verified, the RCL manager is responsible for determining which subset of parameters within the RCL are to be allowed." *Dan* at 5-6. "The RCL enforcer is invoked when the code is executed," and it "monitors accesses by the code to ensure that the permissions and resources consumptions limits are not violated." *Id.* at 6. Indeed, a person having ordinary skill in the art would understand that monitoring permissions and resource consumption limits are examples of security tasks performed to detect malicious software (or malware).

(b) *6(a) – interrupting the loading of a new program for operation with the computing device;*

101. Dan teaches a method for "Installation of Untrusted Applications" where "[t]he application and RCL are downloaded to the host (via network, diskette, CD-ROM, satellite, etc.)." *Dan* at 20. Dan's application is *new* because it is newly downloaded to the host computer. Downloaded applications may also be *new* in the sense that they have not been executed by the client computing device before.

102. A person having ordinary skill in the art would have understood that an application is a computer program that performs a specific task. *See, e.g.*, *Microsoft Computer Dictionary* at 27 (defining "application" as "[a] program designed to assist in the performance of a specific task, such as word processing, accounting, or inventory management."). Indeed, the terms "program" and "application" are often used interchangeably. Thus, a person having ordinary skill in the art would have

understood that Dan's application is a *program* because an application is a type of computer program.

103. For these reasons, a person having ordinary skill in the art would understand that Dan's application is *a new program*.

104. Dan is primarily focused on "unknown programs" that create "an atmosphere of untrust" and "exacerbate[] the need to secure information and physical resources from any alien code. *Dan* at Abstract; 20 (describing the "Installation of Untrusted Applications"). When the computer receives a new program, its resource capability list (RCL) and certificate, a verifier module "verif[ies] that the code and RCL are valid (not tampered with)." *Id.* at 5. Once the code is verified, the RCL manager determines "which subset of parameters within the RCL are to be allowed" and "stores the code and its modified RCL in a secure location." *Id.* at 5-6. "The RCL enforcer is invoked when the code is executed," and it "monitor[s] accesses by the code to ensure that the permissions and resource consumption limits are not violated." *Id.* at 6. The RCL enforcer is "invoked for each system call made by an application." *Id.* at 12. Dan also teaches allowing "downloading (or installing from CDROM) of any code without any certificate as in the conventional systems" and that in these cases, "[s]uch untrusted codes when executed via sandbox environment will be given minimum default resource capabilities." *Id.* at 7.

105. However, a person having ordinary skill in the art would have understood that not all applications, including applications without certificates, are untrusted applications that deserve minimum resource capabilities. An application from a trusted developer, for example, could be considered trusted. Dan's system relies on a third-party certification agency (or "certificate authority") to issue certificates for the code and RCL. *Dan* at 6 ("A code production system communicates with a Certification Agency (CA), which is a trusted third party. The CA issues a certificate for the code and a certificate for the resource list of that code."). However, not all trusted developers communicate with a certification agency to authenticate the code. For example, Dan teaches an exemplary environment for the sandbox system may be a "network computer environment … customized for a specific organization[.]" *Id*. at 2. In such an enterprise environment, the organization may obtain programs directly from a trusted developer and not through the third-party certification agency.

106. It was well known that an enterprise organization may identify a collection of approved applications as trusted on behalf of all its employees. For example, Lambert teaches that "[c]omputer users, and particularly business users, typically rely on a set of programs that are **known and trusted**." *Lambert* at [0003] (emphasis added). "However, in most enterprise environments, particularly in larger environments in which some flexibility is important with respect to what various

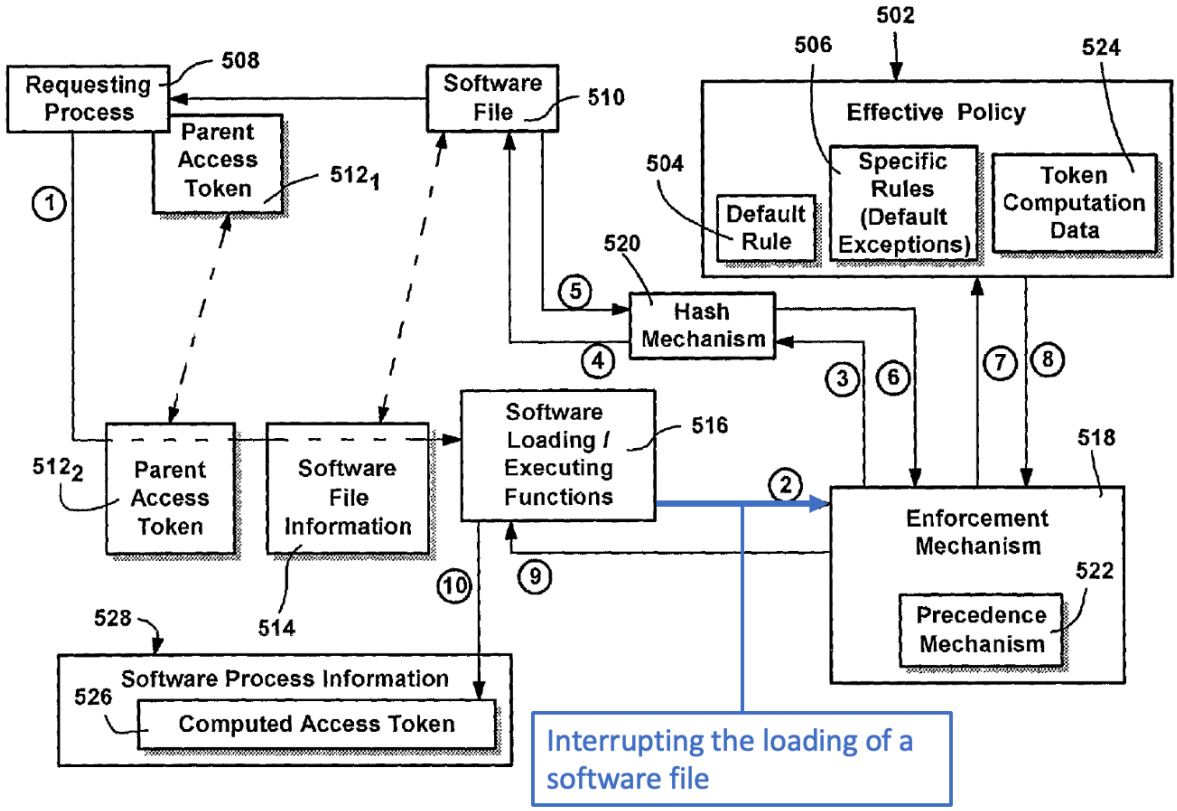
employees need to do with computers to perform their jobs, users are able to run **unknown and/or untrusted code**, that is, programs outside of the supported set." *Id.* To address the presence of trusted and untrusted programs, Lambert teaches applying different "security levels for execution" including "normal (unrestricted), constrained (restricted) or disallowed (revoked)." *Id.* at [0013]. Lambert further teaches that "known good code…is allowed to execute" in either "unrestricted (normal) or restricted" modes while "known bad code…will not be allowed to execute at all." *Id.* at [0056].

107. Acknowledging the existence of "good" and "bad" programs needing varying levels of security, Lambert proposes "a method and system that determines access to resources based on an identification (classification) of software that may contain executable content." *Id.* at [0052]. "To this end, various types of software and specific software files (or other suitable data structures) may be identified and classified, with rules set and maintained for the software classifications that automatically and transparently determine the ability of software to execute." *Id.* "[W]hen software is to be **loaded (e.g., its file opened and some or all thereof read into memory)**, one or more rules are located that correspond to the classification for that software, (or if none corresponds, the default rule), and a selected rules is used **to determine whether the software can be loaded**, and if so, to what extent any processes of the software will be restricted." *Id*. (emphasis added); *see also, id.*, at

[0014] ("In one implementation, **prior to loading a file**, the various functions (such as of an operating system) that enable software to be loaded or otherwise executed are arranged to automatically and transparently communicate with an enforcement mechanism that determines the rule to apply for each file, based on the identification/classification information of the file… Based on located rule, the enforcement mechanism either returns the normal access token, a 'disallowed' error that fails the request, (e.g., the function does not load the software), or computes and returns a restricted access token that restricts the software to an extent that corresponds to the security level determined by the rule.") (emphasis added).

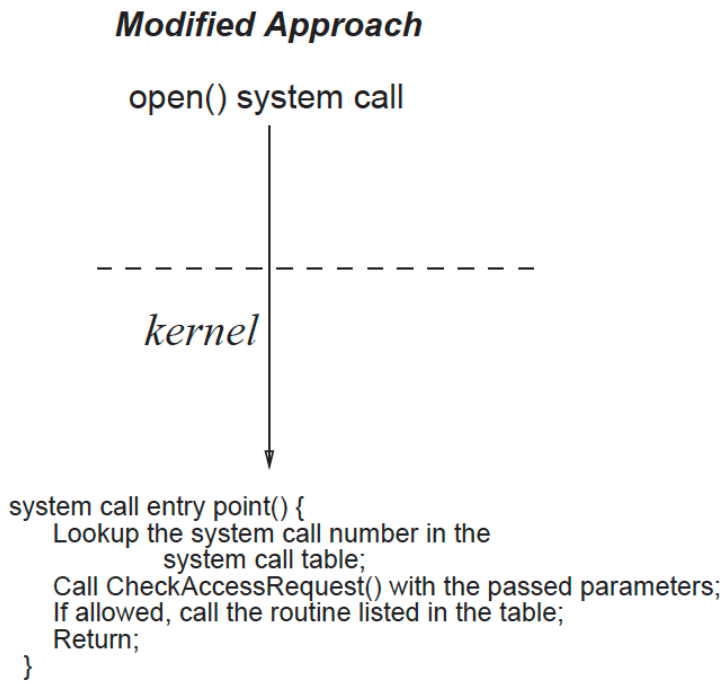
108. This is also depicted in the flowchart of Figure 5A. "When the process 508 **wants to load** and possibly execute the software file 510, it identifies itself and provides information 514 identifying the software file to an appropriate function 516 (of a set) that can open, **load** and/or execute the file 510." *Id.* at [0066] (emphasis added). "**[I]nstead of simply loading the file** and executing any executable code therein with the parent token's rights and privileges, each of the functions 516 is arranged to first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG. 5A)." *Id.* at [0067] (emphasis added). After the loading of the software file is interrupted by function 516 sending the software file information to the enforcement mechanism 518 in step 2, "the enforcement mechanism consults the effective policy 502 to determine which rule

applies for the file 510, and from the rule determines whether to open/execute the file, and if so, the extent of any restricted execution context for the file 510." *Id.* In other words, the loading function 516 is invoked by the requesting process 508, and the loading process is interrupted by the loading function 516 sending the software file information to the enforcement mechanism 518 in step 2 of Figure 5A. Thus, Lambert teaches *interrupting the loading of* the software file so that it can be identified/classified, and an appropriate security policy may be applied:



Id. at Fig. 5A (annotated).

109. In my opinion, a person having ordinary skill in the art would have understood the combination of Dan and Lambert renders obvious the step of *interrupting the loading of a new program for operation with the computing device*. As I discussed above, Dan teaches an RCL enforcer module that "monitors all system calls." *Dan* at 12; *see also* Section III.E.1. Specifically, Dan teaches that "the **operating system's call interface** is modified to call the `CheckAccessRequest()` upon entry to verify the parameters of the system call." *Id.* at 14 (emphasis added). Dan expressly teaches intercepting an "open() system call" by modifying the operating system's call interface to call the `CheckAccessRequest()` routine:



Id. at Fig. 10 (excerpt).

110. This is very similar to Lambert's teaching of the operating system function 516 passing information about the software file to the enforcement mechanism 518 rather than loading the software file. *Lambert* at [0066] ("When the process 508 wants to load and possibly execute the software file 510, it identifies itself and provides information 514 identifying the software file to an appropriate function 516 (of a set) that can open, load and/or execute the file 510. For example, this can be accomplished by placing an application programming interface (API) call to the operating system, wherein the file information 514, typically in the form a path and filename is passed, as a parameter."), [0067] ("[I]nstead of simply loading the file and executing any executable code therein with the parent token's rights and privileges, each of the functions 516 is arranged to first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG. 5A)."), Fig. 5A.

111. Based on the teachings of Lambert, it would have been obvious to modify Dan's RCL enforcer to *interrupt the loading of a new program for operation with the computing device*. As noted above, Lambert teaches classifying a program as "bad" code that is not allowed to execute, "good" code that is allowed to execute without restriction, or "good" yet still untrusted code that is allowed to execute with restrictions. *Lambert* at [0042] ("A process 300 may be associated with the restricted token 210…such as when the process is **untrusted**[.]") (emphasis added), [0056]

("In the case of known **bad** code, the administrator may set the rule to specify that any file with that hash will not be allowed to execute at all… Alternatively, in the case of known **good** code that is allowed to execute, a hash rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent).") (emphasis added).

112. A person having ordinary skill in the art would have understood that configuring Dan's RCL enforcer to interrupt the loading of a new application so that it can be identified and classified prior to execution would have had multiple benefits.

113. First, this modification would have enabled Dan's RCL enforcer to block known malware from executing. As I discussed above, Dan allows "downloading (or installing from CDROM) of any code without any certificate as in the conventional systems." A user may accidentally (or intentionally) download known malware to the computer that does not have a certificate or RCL. Configuring Dan's RCL enforcer to interrupt the loading of the new program would have enabled the RCL enforcer to identify the program as "bad" code prior to it executing and potentially harming the computing device.

114. A person having ordinary skill in the art would have recognized making a security determination prior to loading the application would reduce the amount of potential damage that could be done by a malicious program compared to if the

malicious program was allowed to execute – even if the malicious program only had access to minimum resource capabilities. It was well understood (and common sense) that running malicious code was harmful to computer systems. *See e.g., Lambert* at [0004]. ("[S]ome of the untrusted code may be malicious code (e.g., a computer virus possibly in the form of a Trojan horse) that intentionally misrepresents itself to trick users into running it, typically inflicting damage, corrupting data or otherwise causing mischief to the computers on which the malicious code runs."). Thus, configuring Dan's RCL enforcer to interrupt the loading of a new application so that it may be identified as known as "bad" code, as taught by Lambert, would have allowed Dan's sandbox to block the execution of known malware.

115. Second, this modification would provide solutions for validating conventional programs that lack Dan's certifications/RCLs. As I discussed above, a person having ordinary skill would have understood that a program may be obtained directly from a trusted developer instead of a third-party certification agency. Incorporating Lambert's identification/classification process would have allowed Dan's sandbox to select an appropriate security policy rather than unnecessarily restricting these trusted programs to one-size-fits-all "minimum default resource capabilities" because they do not have certificates/RCLs. *Dan* at 7 ("Finally, CV allows downloading (or installing from CDROM) of any code without any certificate

as in the conventional systems. Such untrusted codes when executed via the sandbox environment will be given minimum default resource capabilities.").

116. Third, configuring Dan's RCL enforcer to interrupt the loading of a new application so that may be identified as either a trusted application that is allowed to run without restrictions or an untrusted application that is allowed to run with restrictions would have also been beneficial. As demonstrated by Lambert, it was well-known that computer users execute *both* trusted and untrusted programs. For example, a program obtained directly from a trusted developer may be an example of a known "good" application that is trusted to execute without restriction. On the other hand, a web browser is an example of a known "good" program that is nevertheless untrusted because it is vulnerable to malicious exploits. *See e.g., De Paoli* at 2 ("[T]he host computer is open to a variety of attacks, ranging from attacks that simply monitor the environment to export information, to attacks that change the configuration or behavior of the host (changing files, consuming resources), and finally to attacks that open a back door to let intruders get into the host. Attacks succeed either because the implementation of the browser is weak (from flaws in the specification and/or from poor implementation) or because the environment in which the browser is executed has flaws.").

117. Configuring Dan's RCL enforcer to interrupt the loading of the new application to allow it to determine whether the new application is a trusted

application that does not need to be monitored by the RCL enforcer or an untrusted application that does need to be monitored by the RCL enforcer would have allowed Dan's sandbox to utilize the computing device's resources more efficiently. Indeed, Dan's RCL enforcer is "invoked for each system call made by an application." *Dan* at 12. It was understood that system call interception was costly and impacted the performance of kernel monitoring systems. *See e.g., Bernaschi* at 175 ("This reduces the cost of system call interception since the invocation of most system calls is not checked, and thus improves the performance of the present prototype."). Not invoking the RCL enforcer for the system calls of a trusted application would have predictably increased the sandbox's performance and would have reduced the computational costs associated with running the sandbox on the client system. In 2002, processors were less powerful than they are today. *See, e.g., Microprocessor Trend Data* (depicting transistors, performance frequency, power, and logical core at 1970-2020). This means that in 2002, a person having ordinary skill in the art would have been motivated to make Dan's sandbox as efficient as possible to ensure that it does not unnecessarily consume resources by monitoring programs that need not be monitored.

118. For these reasons, a person having ordinary skill in the art would have been motivated to improve similar computer security methods in the same way by configuring Dan's RCL enforcer to interrupt the loading of a new application on the

computing device so that it can be identified/classified prior to execution, as taught by Lambert. As discussed above, this would have predictably provided additional safeguards and performance improvements to Dan's client sandbox.

119. Configuring Dan's RCL enforcer to interrupt the loading of the new application so that it can be classified prior to execution would not have changed the principle of operation of Dan's client sandbox. Indeed, Dan's sandbox would still monitor untrusted applications, as described by Dan. The only change would be that known malicious applications would be blocked from execution, and trusted applications would not be monitored. Given this and the predictability of computing programming techniques at the time of the '137 Patent and Dan's teachings regarding modifying the operating system's call interface, there would have been a reasonable expectation of success configuring Dan's RCL enforcer to *interrupt the loading of a new program for operation with the computing device*.

(c) *6(b) – validating the new program;*

120. Regarding the *validating* step, the '137 Patent discloses "the validation module can represent the **MD5 software module** commonly known to those in the art," which "calculates a checksum for each allowed program and that checksum is stored for later comparison." *'137 Patent* at 8:13-17.

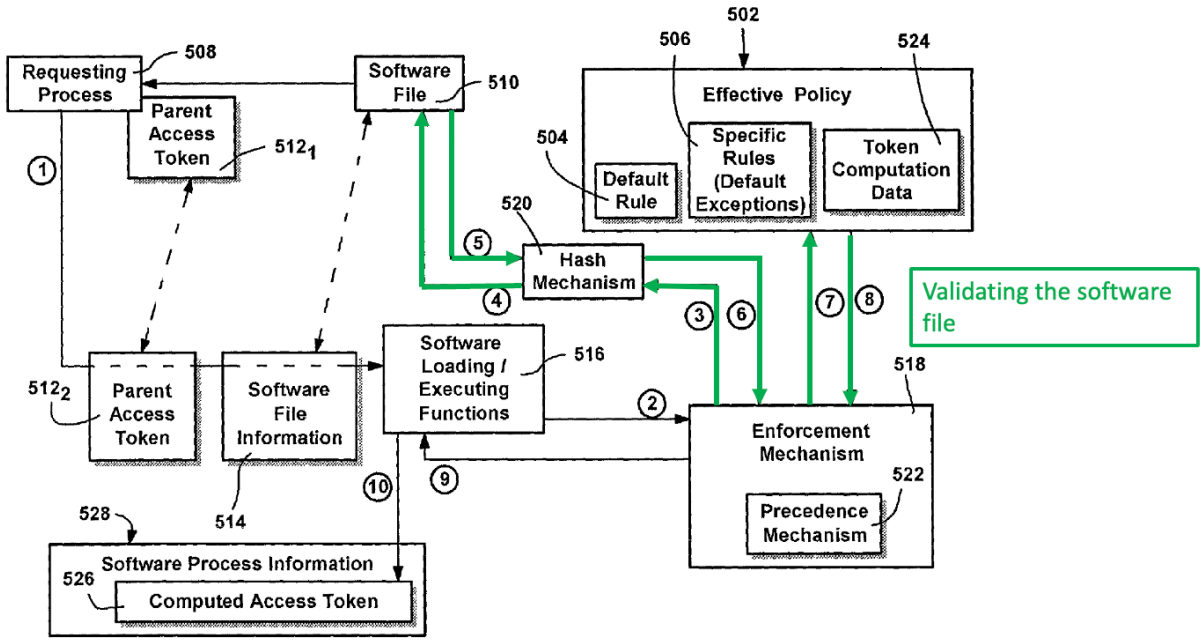
121. As I described above, it would have been obvious to a person having ordinary skill in the art to configure Dan's RCL enforcer to interrupt the loading of

the new application (i.e., *the new program*) so that that application may be identified and classified, and an appropriate security policy may be selected as taught by Lambert. *See* Claim 6(a). Lambert's identification/classification process is very similar to the '137 Patent's *validating* step. Specifically, Lambert teaches that "[t]o identify and classify software, (e.g., maintained in a file), various selectable classification criteria are made available to an administrator, including **a unique hash value of the file's content (e.g., via an MD5 image hash**)…" *Lambert* at [0053] (emphasis added). Per Lambert, "one way that a software file can be identified (and thus matched to a corresponding rule) is by **a hash of its contents**, which is essentially a fingerprint that uniquely identifies a file regardless of whether the file is moved or renamed." *Id.* at [0055] (emphasis added). "[B]efore opening a file for execution, **a hash value is determined from that file's contents**, and the set of hash value information (e.g., in the policy object) searched to determine if a rule exists for that hash value. If so, that rule provides the security level for the software." *Id.* (emphasis added). Thus, like the '137 Patent, Lambert also teaches comparing the hash of the software file to the hashes of known programs.

122. Per Lambert, "an administrator can set a rule for any file based on its unique hash value, with the rule specifying whether (and if so, how) a software file having that hash value will execute." *Id.* at [0056]. "In the case of known bad code, the administrator may set the rule to specify that any file with that hash will not be

allowed to execute at all (i.e., 'disallowed' or 'revoked' status)." *Id.* "Since the rule along with the hash value is maintained or distributed to the machine, (e.g., via a policy object), whenever a request to open that file is received, a hash of the file matches the saved hash and rule, and that file will not be run." *Id.* "Alternatively, in the case of known good code that is allowed to execute, a hash rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent)." *Id.* Thus, Lambert compares the software file's hash to the hashes of known programs to validate whether it should be allowed to execute in an unrestricted manner, restricted manner, or not at all.

123. Lambert's *validating* step is also depicted in Figure 5A. Once the enforcement mechanism receives the software file's information from the software loading function 516, it "may call (or include in its own code) a hash function 520 or the like to obtain a hash value from the contents of the software file 510, as generally represented in FIG. 5A by the arrows labeled with circled numerals three (3) to six (6)." *Id.* at [0068]. "As described above, a suitable hash function 520 provides a unique fingerprint of the software file 510, while the effective policy 502 may have a specific rule for that hash." *Id.* Arrows seven (7) and eight (8) "represent[] accessing the policy to get the rule or rules" by looking for a rule corresponding to "the hash value." *Id.* Therefore, steps 3-8 of Figure 5A collectively qualify as *validating* the software within the meaning of the '137 Patent:



Id. at Fig. 5A (annotated).

124. For all the reasons discussed above, it would have been obvious to person having ordinary skill in the art to configure Dan's sandbox to identify and classify applications as bad (i.e., no execution), good (unrestricted execution), or untrusted (restricted execution) as taught by Lambert. *See* Claim 6(a). Since Lambert's identification/classification step qualifies as *validating,* Dan's sandbox modified in view of Lambert also renders obvious *validating the new program* for the same reasons. *Id.*

125. Moreover, it would have been obvious to a person having ordinary skill in the art to configure Dan's RCL enforcer to validate the new application by comparing its hash value to a list of hashes for known applications so that the

sandbox can distinguish between trusted and untrusted applications, as taught by Lambert. A person having ordinary skill in the art would have understood that using a hash of the new application and comparing it to a list of hashes for known applications would have been a reliable way to validate the application. As noted by Lambert, the hash serves as "a fingerprint that uniquely identifies a file regardless of whether the file is moved or renamed." *Lambert* at [0055]. For these reasons, a person having ordinary skill in the art would have been motivated to combine the prior art elements of Dan's RCL enforcer and Lambert's method of using hashes to uniquely identify and classify a software file according to known methods to yield the predictable result of allowing Dan's sandbox to validate the new application by identifying it as bad (i.e., no execution), good (unrestricted execution), or untrusted (restricted execution).

126. The '137 Patent acknowledges that the MD5 software modules were "commonly known to those in the art." *'137 Patent* at 8:14-15. A person having ordinary skill in the art would have understood that an MD5 hash was a well-known type of cryptographic hash. *See, e.g.*, *Preneel* at 20-21 (listing MD5 as a type of dedicated cryptographic hash function). Therefore, there would have been a reasonable expectation of success configuring Dan's sandbox to validate the new application using "commonly known" MD5 hashes.

(d) *6(c) – if the new program is validated, permitting the new program to continue loading and to execute in connection with the computing device;*

127. I have been asked to interpret this limitation as "if the new program is validated, then it is not monitored while it loads and executes in connection with the computing device." *See* Section III.D.

128. As I described above, Lambert teaches comparing the hash of a software file to the hashes of known software files. *See* Claim 6(b). Lambert also teaches a case where the software file is determined to be "good code" that is allowed "unrestricted" execution based on this comparison. *Lambert* at [0056] ("[I]n the case of known good code that is allowed to execute, a hash rule can specify how it will be executed, e.g., unrestricted (normal)…"); *see also, id.* at [0008] ("Depending on what the rule specifies for the classification that is determined for any given software module (e.g., software file), the software may execute in a normal (unrestricted) execution environment (context)…").

129. Unrestricted execution is different from restricted execution in that restricted execution "reduces software's access to resources, and/or removes privileges, relative to a user's normal access token." *Id.* at Abstract. When it is determined that software should be restricted, "the enforcement mechanism 518 can compute and associate a restricted token with the software file 510, whereby if creation of a process is requested for the software file, (it includes executable

content), any process of the software file is restricted in its access rights and privileges according to the restricted token." *Id.* at [0070]. When the user requests to load a software file, "the enforcement mechanism 518 completes the call and returns information back to the software function 516 that called it, the returned information essentially including an instruction on whether to open the file, and if so, the token to associate with the file." *Id.* at [0071]. "If a token is returned, it may be the parent token (unchanged) if no restricted execution environment is required by the rule, or a restricted token that establishes a restricted execution environment/context for the processes of the software file 510." *Id.* "If the computed token 526 is a restricted token, the process of the software file can now execute, but only in association with the restricted token, thus providing the rule-determined secure execution environment." *Id.*

130. As I discussed above, Lambert's Figure 3 and the related disclosure describe how a restricted program is monitored based on its restricted token. *See* Section III.E.2. Specifically, every time a restricted program attempts to access an object (e.g., a file object), the access must be approved or denied based on the restrictions in the program's restricted token. *Id.*

131. In contrast, Lambert teaches that unrestricted code is not monitored or restricted. *Lambert* at [0008] ("[T]he software may execute in a normal (unrestricted) execution environment (context)…"), [0098] ("If not disallowed, step

904 tests whether the security level is unrestricted. If so, the enforcement mechanism effectively returns the parent access token (e.g., the token itself, a pointer to it, or permission to use the parent access token), possibly along with a return code having a value indicative of success." In other words, Lambert teaches that monitoring is only required when a program has a restricted token. Thus, Lambert teaches determining that the software is validated for unrestricted execution, which results in it not being monitored while it loads and executes on the computing device.

132. For the reasons discussed above, it would have been obvious to a person having ordinary skill in the art to configure Dan's RCL enforcer to validate the new program by comparing its hash to the hashes of other known programs. *See* Claim 6(b). For the same reasons, it would have been obvious to configure Dan's RCL enforcer to determine that *the program is validated* if its hash matches the hash of a known trusted application. *Id.* As I discussed above, it would have been obvious to configure Dan's sandbox to determine whether the new application is a trusted application that does not need to be monitored by the RCL enforcer to allow Dan's sandbox to utilize the computing device's resources more efficiently. *See* Claim 6(a). As I noted above, a person having ordinary skill in the art would have been motivated to make Dan's sandbox as efficient as possible to ensure that it does not unnecessarily consume resources by monitoring programs that need not be monitored. *Id.* This was especially critical in 2002 because processor speeds much

slower than today. *Id.* In my opinion, a person having ordinary skill in the art would have understood that a trusted application would not pose a serious security risk to the client system. Therefore, monitoring a trusted application would be a waste of the computing device's resources. Therefore, a person having ordinary skill in the art would have been motivated to improve similar computer-implemented security methods in the same way by configuring Dan's RCL enforcer to not monitor a trusted application during loading and execution.

133. Configuring Dan's RCL enforcer to not monitor trusted programs would have represented a minor change to the sandbox software. Modifying Dan's RCL enforcer to not monitor trusted applications would also not have changed the principle of operation because Dan would still monitor untrusted applications per Dan's stated purpose. *Dan* at Abstract ("Sharing of unknown programs creates an atmosphere of untrust and hence exacerbates the need to secure information and physical resources from alien code. A sandbox approach to execution of such foreign code is proposed in this paper."). Dan also proposes an embodiment where only programs with trusted certificates execute and they are not monitored, which would have further encouraged or suggested to a skilled artisan to not monitor when a certificateless program was identified for unrestricted execution using Lambert's features. *Id.* at 2 ("allowing execution…without further monitoring"). Given these factors and the predicably of computer programming at the time of the '137 Patent,

a person having ordinary skill in the art would have had a reasonable expectation of success configuring Dan's RCL enforcer to not monitor a trusted application while it is loading and executing on the client system.

(e) *6(d)(i) – if the new program is not validated, monitoring the new program while it loads and executes in connection with the computing device,*

134. Lambert also teaches comparing the hash of a software file to the hashes of known software files and determining that the software must be executed "in a restricted execution environment." *Lambert* at [0008]; *see also, id.*, at [0014] ("Based on located rule, the enforcement mechanism … computes and returns a restricted access token that restricts the software to an extent that corresponds to the security level determined by the rule. If allowed but restricted, the returned restricted token is associated with the code's processes, constraining the code accordingly."), [0054] ("By enabling the identification of software via these or other classification criteria, and then locating and enforcing an appropriate security rule established for the software based on its classification, the present invention automatically controls whether content is executable, and if so, the extent (if any) to which it is restricted."). As discussed above, Lambert teaches that applications executing in a restricted execution environment are monitored. *See* Claim 6(c).

135. Dan's RCL enforcer monitors the new program while it loads and executes on the computing device. Specifically, the RCL enforcer "ensures that the

permissions and resource consumption limits specified in the RCL are not violated." *Dan* at 9. "The module for enforcing system resource capabilities monitors two types of resources: physical and logical." *Id.* Regarding the monitoring of physical resources, the "RCL enforcer protects physical resources such as memory and CPU usage, disk storage, and disk bandwidth utilization." *Id.* "Access control information about physical and logical resources of the processing executing an alien application code is maintained in data structures called the Runtime Physical Resources Table (RPRT) and the Runtime Logical Resources Table (RLRT)." *Id.* "**When an application is initially loaded for execution**, the RPRT and RLRT data structures are initiated from the information specified in the RCL configuration file associated with this application." *Id.* (emphasis added). "**During process execution**, the RCL enforcer references and updates the information stored in the RPRT data structure using the algorithm outlined in Figure 5 to allow or deny the request for accessing physical resources." *Id.* (emphasis added).

136. "Similar to the RPRT, access control information for the logical resources is maintained in a data structure called the Runtime Logical Resources Table (RLRT)." *Id*. at 12. "The logical resources constitute file, directories, communication ports and various other system service calls." *Id.* "The RLRT data structure is also initialized **when an application is loaded into memory**." *Id.* (emphasis added). "The RLRT contains a row for each system service access

capabilities." *Id.* "Each row has a flag that indicates whether this service call is allowed always, never or needs to be verified[.]" *Id.* "The RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system." *Id.* "This monitoring requires that the RCL enforcer be invoked for each system call made by an application." *Id.* "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT data structures and either allows or disallows the call from continuing." *Id.* Thus, Dan teaches *monitoring the new program while it loads and executes in connection with the computing device*.

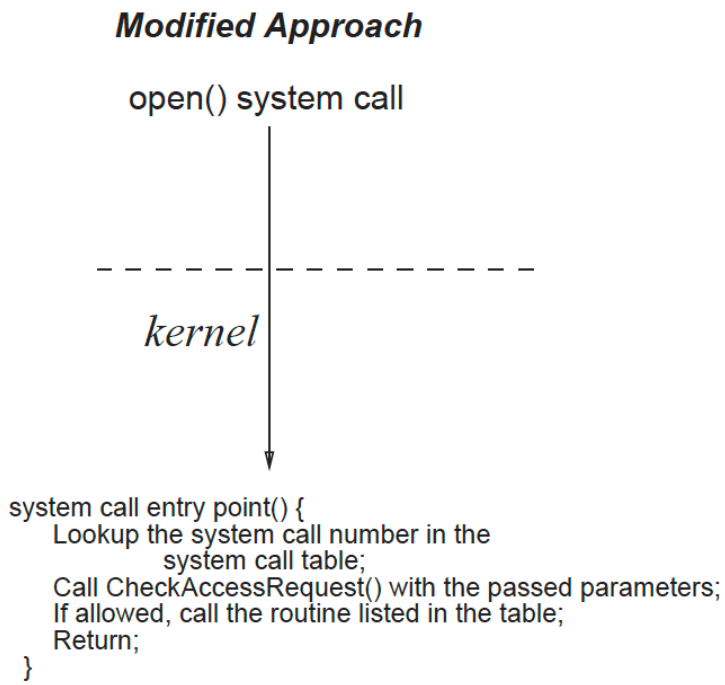
137. When modifying Dan in view of Lambert, it would have been obvious to a person having ordinary skill in the art to configure Dan's sandbox to perform monitoring of the new program while it loads and executes on the host computer when the program fails validation for unrestricted execution, including for programs that lack Dan's certificate. A person having ordinary skill in the art would have understood this modification simply amounts to making the monitoring conditional on the validation outcome and to not perform monitoring on new programs that are determined to be trusted, as discussed with regard to Claim 6(c), while monitoring non-validated programs with an appropriately configured RCL similar to how Lambert customizes monitoring by restricted tokens. *Lambert* at [0037]. As explained above, Dan teaches monitoring untrusted new programs while they load and execute on the client device. It would have been obvious to configure the

sandbox to perform monitoring on new applications as described by Dan, based on the outcome of the validation. Thus, a person having ordinary skill in the art would have been motivated to combine prior art elements according to known methods to yield the predictable result Dan's sandbox monitoring the untrusted application when it fails the validity check. Since the purpose of Dan's sandbox is to monitor untrusted applications, and this is a highly predictable art, there would have been a reasonable expectation of success configuring Dan's sandbox to monitor the new application when it fails the validity check.

(f) *6(d)(ii) – wherein the step of monitoring the new program while it executes is performed at the operating system kernel of the computing device.*

138. Dan teaches that "[t]he RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system." *Dan* at 12. As discussed above, Dan teaches three different approaches to monitoring system calls, including binary code modification, service library modification, and system call modification. *Id* at 14; *see also*, Section III.E.1. In Dan's preferred system call modification approach, "the **operating system's call interface** is modified to call the `CheckAccessRequest()` upon entry to verify the parameters of the system call." *Dan* at 14 (emphasis added). "Thus, regardless of how the applications are structured, **the system call entry point in the operation systems** will ensure consistent security checks for all applications that are executed on the machine." *Id.*

(emphasis added). As shown in Figure 10, the system call modification technique performs monitoring at the operating system kernel of the client system:



Id. at Fig. 10 (excerpt).

139. A person having ordinary skill in the art would have understood the portion of Figure 10 reproduced above shows that when the operating system kernel system call function is invoked, the enforcer module's `CheckAccessRequest()` routine determines whether the call is allowed. *Id.* at 13 (depicting pseudocode for the `CheckAccessRequest()` routine). Indeed, a person having ordinary skill in the art would have understood, that the new program is identified by the system call number using the call table, and is monitored by means of analyzing the passed parameters and the identity of the new program, after which a conditional action is

taken based on the determination (i.e., the system call is either allowed or disallowed). *Id.* Thus, Dan teaches that *the step of monitoring the new program while it executes is performed at the operating system kernel of the computing device*.

140. A person having ordinary skill in the art would have found it obvious to implement Dan's sandbox using the expressly suggested system call modification method. Dan expressly recognizes that using the system call modification method is preferred because it would have ensured consistent security checks for all untrusted applications that are executed on the client system. *Dan* at 14 ("Thus, regardless of how the applications are structured, the system call entry point in the operation systems will ensure consistent security checks for all applications that are executed on the machine."); 15 ("We selected this approach to manage logical resources in the system primarily because of the consistency it offers for enforcing security."). Given Dan's express teaching, suggestion, and motivation, a person having ordinary skill in the art would have been motivated to implement Dan's sandbox using the system call modification method.

141. As discussed above, Dan teaches the system call modification method is the most consistent way to enforce security. Thus, a skilled artisan would have understood that there would have been a reasonable expectation of success configuring Dan's sandbox to use the system call modification method since it was the most consistent way to enforce security.

2. Claim 8 - The method of claim 6, wherein the step of monitoring the new program comprises intercepting a signal from the computing device's operating system.

142. As I described above, Dan teaches monitoring the new application by *intercepting a signal from the computing device's operating system* by modifying the operating system's call interface to call the RCL enforcer module's `CheckAccessRequest()` routine. *See* Claim 6(d)(ii).

3. Claim 9 - The method of claim 6, wherein the step of validating the new program comprises determining whether the new program corresponds with an approved program.

143. As I discussed above, it would have been obvious to a person having ordinary skill in the art to configure Dan to incorporate Lambert's validating step to identify and classify the new program. *See* Claim 6(b). As part of this step, Lambert teaches comparing the hash of a software file to the hashes of other known software files to determine if the software file being loaded corresponds to an approved program that is allowed to execute in unrestricted mode. *Id.* Thus, a person having ordinary skill in the art would have understood the combination of Dan and Lambert also teaches *wherein the step of validating the new program comprises determining whether the new program corresponds with an approved program* for the same reasons as discussed with regard to Claim 6(b). *Id.*

4. Claim 12 - The method of claim 6, wherein the step of monitoring the new program comprises controlling the files the new program attempts to access during execution of the new program.

144. Dan teaches that "[t]he RCL enforcer monitors all systems calls, i.e., access requests to logical resources in the system." *Dan* at 12. Dan describes, "logical resources constitute **file**, directories, communication ports and various other system service calls." *Id*. (emphasis added); *see also, id.* at 17 ("Following this syntax specification, a list of comma-separated entries are described which specify the application privileges for an **open call to files in various directories**. The example shows that the open call should be allowed in any mode for **all files** in the `/tmp` directory and not be allowed for any **files** in the `/tmp/admin` subdirectory.") (emphasis added). After the RCL enforcer is invoked by a system call attempting to open a file, "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT data structures and either allows or disallows the call from continuing." *Id.* at 12.

145. For this reason, a person having ordinary skill in the art would have understood that Dan teaches *the step of monitoring the new program comprises controlling the files the new program attempts to access during execution of the new program*.

5. Claim 25

(a) *25[Pre]. – A computer-implemented method for performing security for a computer device during a pre-execution phrase comprising the steps of:*[2]

[2] I have been informed that the preamble of claim 25 includes a typographical error and that the term "phrase" should be interpreted to mean "phase."

146. As I discussed above in Claim 6, Dan teaches *a computer-implemented method for performing security for a computer device*. *See* Claim 6[Pre]. Additionally, the combination of Dan and Lambert teaches *a computer-implemented method for performing security for a computer device during a pre-execution phase* as described below.

(b) *25(a) – identifying an allowed program that is permitted to execute with the computing device;*

147. As I discussed above, Dan primarily focuses on unknown/untrusted programs and automatically assumes that any program without a certificate/RCL should be given with minimum resource capabilities. *See* Claim 6(a). However, a person having ordinary skill in the art would have understood that not all applications without certificates are unknown and/or untrusted. *Id.*

148. Lambert teaches a step where the system administrator identifies a software file as being allowed to execute on a computing device in either a restricted or unrestricted mode. "Via a user interface or the like operably connected to a suitable hash mechanism, an administrator can select any software file, obtain a hash value from its contents, and enter and associate a rule (that specifies a security level) with that hash value." *Id* at [0055]. "Multiple files can have their corresponding hash values and rules maintained in a policy object, or otherwise made available to the system, whereby for any given instance of any file, the set of maintained hash values can be queried to determine whether a rule exists for it." *Id.* "In this manner, an

administrator can set a rule for any file based on its unique hash value, with the rule specifying whether (and if so, how) a software file having that hash value will execute." *Id* at [0056]. "In the case of known bad code, the administrator may set the rule to specify that any file with that hash will not be allowed to execute at all (i.e., 'disallowed' or 'revoked' status)." *Id.* "[I]n in the case of known good code that is **allowed to execute**, a hash rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent)." *Id.* (emphasis added). Thus, Lambert's teaching of an administrator identifying a "good" program that is allowed to execute in either restricted or unrestricted mode qualifies as *identifying an allowed program that is permitted to execute with the computing device*.

149. Based on the teachings of Lambert, it would have been obvious to configure Dan's system to include a user interface that allows the system administrator to *identify an allowed program that is permitted to execute with the computing device*. Specifically, it would have been obvious to a PHOSITA to configure Dan's system to allow the system administrator to identify "good" programs that are allowed to execute in an unrestricted or less restricted manner even if they do not have certificates/RCLs. Indeed, configuring Dan's system to include a user interface where an administrator can identify allowed applications would have enabled the administrator to define security policies for applications that do not have certificates/RCLs. It was a well-known practice for system administrators to

maintain lists or databases of hashes associated with known applications. *See, e.g.*, *Schmid* 4:36-38 ("To make an administrator's job easier, an AS may maintain a database of known application hashes."). A skilled artisan would have understood that allowing a system administrator of Dan's system to identify allowed programs in advance would have predictably made it easier to ascertain whether a given application should be allowed to be executed and, if so, what (if any) restrictions should apply in the absence of a certificate/RCL. Therefore, person having ordinary skill in the art would have been motivated to combine the prior art elements of Dan's security method with Lambert's step of allowing a system administrator to identify an allowed program that is permitted to execute with the computing device to yield the predictable result of a security system that applies appropriate security policies to applications that do not have certificates/RCLs.

150. Dan teaches providing utilities for the system administrator, including an `InstallCop` utility that the system administrator uses to edit configuration files and manage the system-wide installation of software applications. *Dan* at 20 ("In case of system wide installation (by the super user)…"), ("[T]he `InstallCop` can allow the system administrator to edit the system default configuration file before the application is made available to users."). A person having ordinary skill in the art would have understood that adding another utility for the system administrator to identify programs that are allowed to execute on the computing device would have

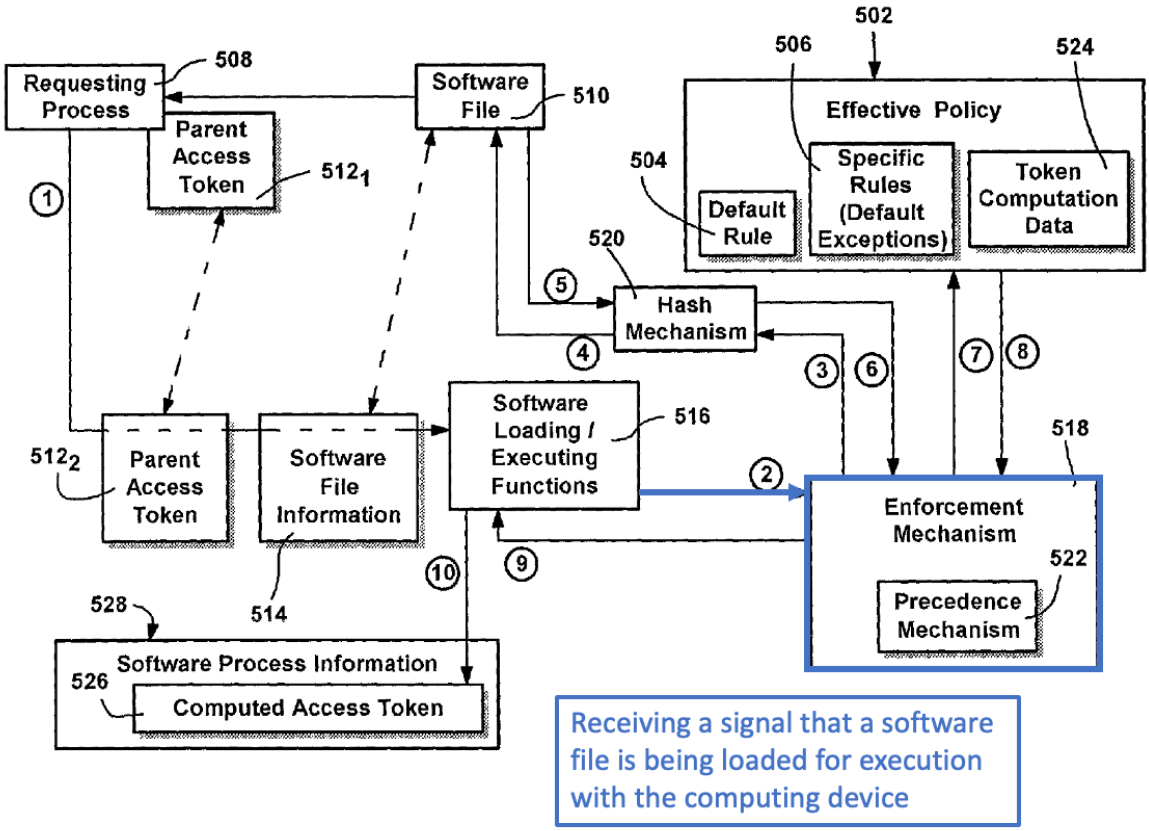
only required minor software changes. Lambert likewise describes administrators configuring hash-based rules by a user interface, further teaching, suggesting, and motivating a person having ordinary skill in the art. *Lambert* at Abstract (describing rules "distributable over a network"), [0010]-[0011], [0052]-[0056], [0078]. Moreover, allowing an administrator to identify applications that are allowed to execute in either unrestricted or restricted mode would not have changed the principle of operation of Dan's sandbox since Dan's sandbox would still execute untrusted code having associated restrictions. Given these factors, there would have been a reasonable expectation of success configuring Dan's sandbox to include a user interface that allows a system administrator to *identify an allowed program that is permitted to execute with the computing device*.

(c) *25(b) – receiving a signal that a new program is being loaded for execution with the computing device;*

151. For the reasons discussed above, Dan teaches *a new program*. *See* Claim 6(a).

152. Lambert teaches *receiving a signal that* a software file *is being loaded for execution with the computing device*. For example, "in FIG. 5A, a process 508 such as a process of a user or application requesting to open a software file 510 has an access token 512_1 associated with it[.]" *Lambert* at [0066]. "**When the process 508 wants to load and possibly execute the software file 510**, it identifies itself and provides information 514 identifying the software file to an appropriate function

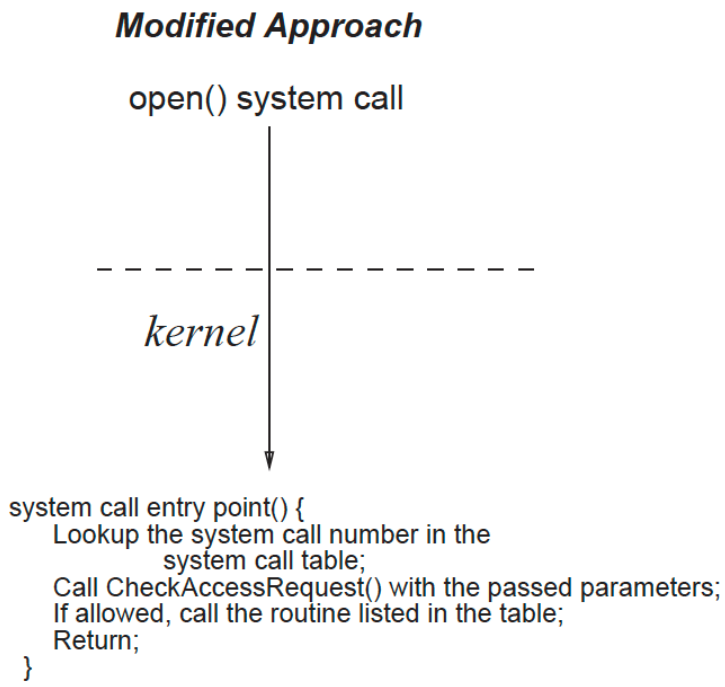
516 (of a set) that can open, **load and/or execute** the file 510." *Id.* (emphasis added). "For example, this can be accomplished by placing an application programming interface (API) call to the operating system, wherein the file information 514, typically in the form a path and filename is passed, as a parameter." *Id.* "Note that in a typical implementation, the software opening function 516 comprises an operating system component, and thus has access to a copy 512_2 of the parent access token." *Id.* "Further note that **such function calls are already the typical way in which files are loaded**, and thus the process 508 (which may be of an existing application program or other type of component) need not be modified in any way." *Id.* (emphasis added). "[I]nstead of simply loading the file and executing any executable code therein with the parent token's rights and privileges, each of the functions 516 is arranged to **first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG. 5A)**." *Id at* [0067]. (emphasis added). Thus, Lambert teaches that the enforcement mechanism 518 *receives a signal that a* software file *is being loaded for execution with the computer device* from the software opening function 516 as shown in Figure 5A below:



Lambert at Fig. 5A (annotated).

153. In my opinion, a person having ordinary skill in the art would have understood the combination of Dan and Lambert renders obvious the step of *receiving a signal that a new program is being loaded for execution with the computing device*. As I discussed above, Dan teaches an RCL enforcer module that "monitors all system calls." *Dan* at 12; *see also* Section III.E.1. Specifically, Dan teaches that "the **operating system's call interface** is modified to call the `CheckAccessRequest()` upon entry to verify the parameters of the system call." *Id.* at 14 (emphasis added). Indeed, Dan expressly teaches intercepting an "open()

system call" by modifying the operating system's call interface to call the `CheckAccessRequest()` routine:



Id. at Fig. 10 (excerpt).

154. This is very similar to Lambert's teaching of the operating system function 516 passing information about the software file to the enforcement mechanism 518 rather than loading the software file. *Lambert* at [0066] ("When the process 508 wants to load and possibly execute the software file 510, it identifies itself and provides information 514 identifying the software file to an appropriate function 516 (of a set) that can open, load and/or execute the file 510. For example, this can be accomplished by placing an application programming interface (API) call to the operating system, wherein the file information 514, typically in the form a

path and filename is passed, as a parameter."), [0067] ("[I]nstead of simply loading the file and executing any executable code therein with the parent token's rights and privileges, each of the functions 516 is arranged to first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG. 5A)."), Fig. 5A.

155. Based on the teachings of Lambert, it would have been obvious to a person having ordinary skill in the art to configure Dan's RCL enforcer module to *receive a signal that a new application is being loaded for execution with the computing device* via the system call functions associated with loading and executing a new program. For the reasons described above, it would have been obvious to a person having ordinary skill in the art to configure Dan's system to allow an administrator to identify applications that are allowed to execute on the computer and define their corresponding security policies to account for applications that do not have certificates/RCLs. *See* Claim 25(a). For the same reasons, it would have also been obvious to configure the RCL enforcer to receive a signal when the new application is being loaded so that the RCL enforcer may check to see if the application is an allowed application and what security policy should apply. Indeed, a person having ordinary skill in the art would have understood that configuring Dan's RLC enforcer to receive a signal (e.g., via modifying the operating system's call interface functions responsible for loading programs) would have predictably

allowed the sandbox to determine whether an application without a certificate is allowed to execute and if so, whether it may execute in a restricted or unrestricted manner, prior to loading it. Therefore, a skilled artisan would have been motivated to combine prior art elements according to known methods to yield the predictable result of allowing Dan's sandbox to determine whether the new application needs to be monitored prior to it being loaded and executed on the computing device even if it does not have a certificate/RCL.

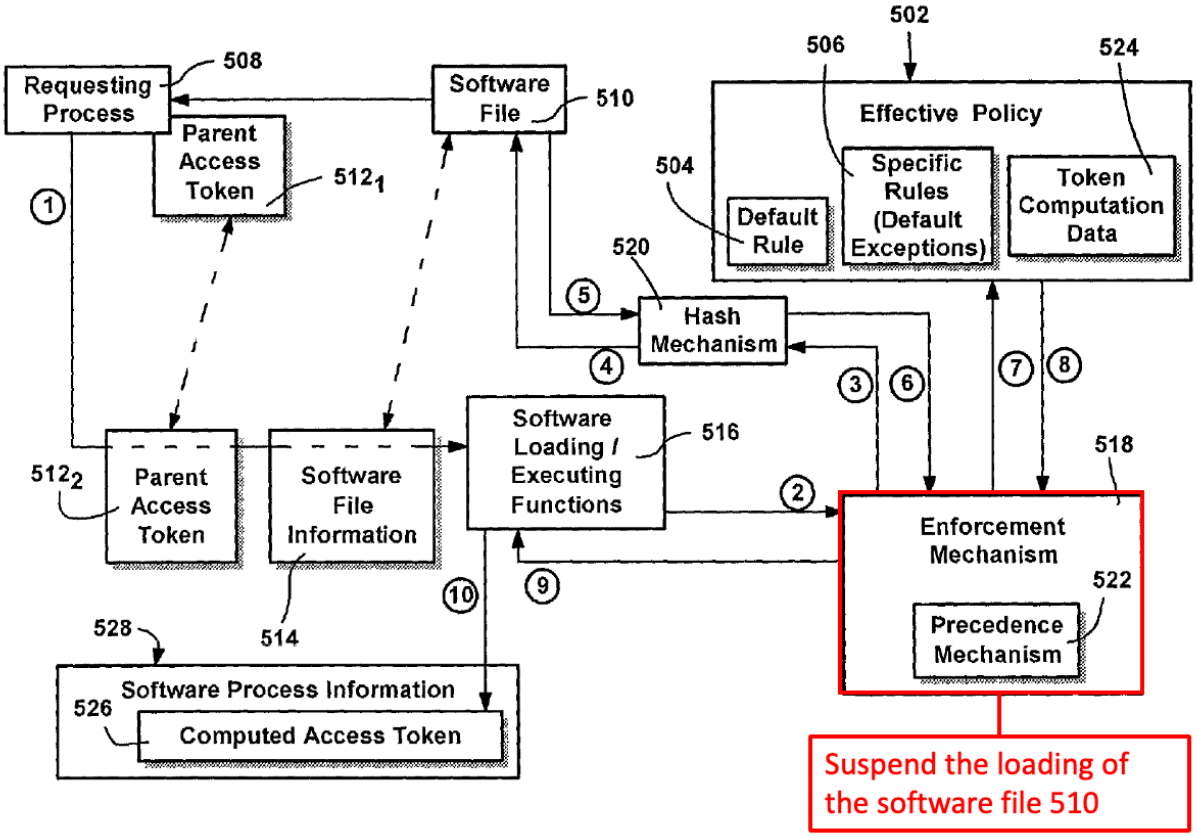
156. Since Dan's RCL enforcer module is already configured to monitor "all system calls," there would have been a reasonable expectation of success configuring Dan's RCL enforcer to receive a signal that a new application is being loaded for execution with the computing device from the operating system call functions associating with loading and executing a new program, as taught by Lambert. *Dan* at 12.

(d) *25(c) – suspending the loading of the new program;*

157. Lambert teaches that the enforcement mechanism *suspends the loading* of the software file. Specifically, Lambert teaches that "when software is to be **loaded (e.g., its file opened and some or all thereof read into memory)**, one or more rules are located that correspond to the classification for that software, (or if none corresponds, the default rule), and a selected rules is used **to determine whether the software can be loaded**, and if so, to what extent any processes of the

software will be restricted." *Lambert* at [0052] (emphasis added); *see also, id* at [0014] ("In one implementation, **prior to loading a file**, the various functions (such as of an operating system) that enable software to be loaded or otherwise executed are arranged to automatically and transparently communicate with an enforcement mechanism that determines the rule to apply for each file, based on the identification/classification information of the file…Based on located rule, the enforcement mechanism either returns the normal access token, a 'disallowed' error that fails the request, (e.g., the function does not load the software), or computes and returns a restricted access token that restricts the software to an extent that corresponds to the security level determined by the rule.") (emphasis added).

158. Referring again to Figure 5A, once the enforcement mechanism 518 receives the software file information 514 from the software loading/execution functions 516, "the enforcement mechanism consults the effective policy 502 to determine which rule applies for the file 510, and from the rule determines whether to open/execute the file, and if so, the extent of any restricted execution context for the file 510." *Id* at [0067].



Id. at Fig. 5A (annotated).

159. Thus, Lambert teaches *suspending the loading* of the software file so that the enforcement mechanism 518 can determine whether the loading of the software is allowed by the rules.

160. Based on the teachings of Lambert, it would have been obvious to a person having ordinary skill in the art to modify Dan's RCL enforcer to *suspend the loading of a new program* to identify and classify it as either a "bad" program that should not be allowed to execute or a "good" program that is allowed to execute in either unrestricted or restricted mode. For the same reasons as discussed above, configuring Dan's sandbox to be able to identify and classify a new program would

have allowed it to select the appropriate security policy even if it does not have a certificate/RCL. *See* Claim 25(b). Thus, a person having ordinary skill in the art would have been motivated to improve similar computer-implemented security methods in the same way by configuring Dan's RLC enforcer to suspend the loading of the new application until the RLC enforcer identifies the application and the security rules that apply to it.

161. As described above, Dan's RCL enforcer module is already configured to monitor "all system calls." *Dan* at 12. As shown in Dan's Figure 10, the system call is not completed unless allowed by the `CheckAccessRequest()` routine, which shows that the modified approach would suspend the call. *Dan* at Fig. 10. Thus, a person having ordinary skill in the art would have had a reasonable expectation of success configuring Dan's RCL enforcer to suspend the loading of the new program upon receipt of a signal from the operating system call interface to allow the RCL enforcer to determine the appropriate security policy for the application even if it does not have a certificate/RCL.

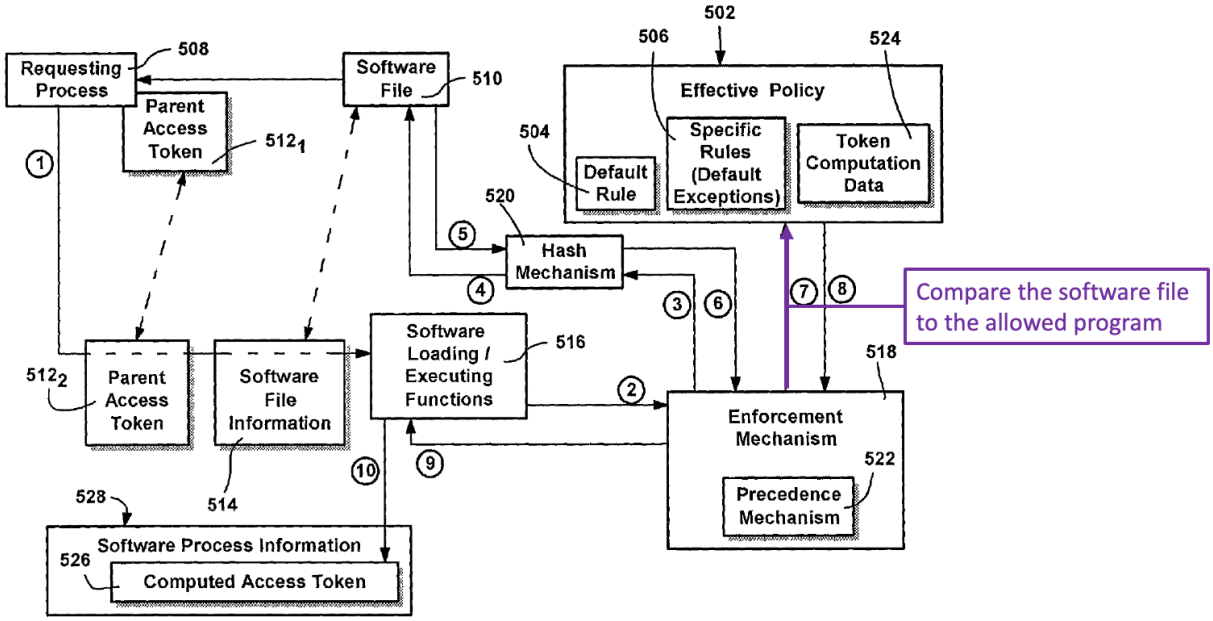
(e) *25(d) – comparing the new program to the allowed program; and*

162. Lambert teaches that the enforcement mechanism *compares the* software file *to the allowed program*. The enforcement mechanism 518 "consults the effective policy 502 to determine which rule applies for the file 510, and from the rule determines whether to open/execute the file, and if so, the extent of any

restricted execution context for the file 510." *Lambert* at [0067]. "More particularly, the enforcement mechanism 518 may call (or include in its own code) a hash function 520 or the like to obtain a hash value from the contents of the software file 510, as generally represented in FIG. 5A by the arrows labeled with circled numerals three (3) to six (6)." *Id.* at [0068]; *see also, id.,* at Fig. 5A. "[A] suitable hash function 520 provides a unique fingerprint of the software file 510, while the effective policy 502 may have a specific rule for that hash." *Id.* For example, Lambert teaches that the hash is "a unique hash value of the file's content (e.g., via an MD5 image hash)." *Id.* at [0053].

163. To determine if the software file's hash matches the hash of a known software file, "the set of maintained hash values can be queried to determine whether a rule exists for it." *Id.* at [0055]. "In other words, before opening a file for execution, a hash value is determined from that file's contents, and the set of hash value information (e.g., in the policy object) [is] searched to determine if a rule exists for that hash value." *Id.* The set of maintained hash values includes the hash value of the "known good code that is allowed to execute" (i.e., *the allowed program*) that was previously identified by the system administrator. *See* Claim 25(a). "Note that while FIG. 5A essentially represents accessing the policy to get the rule or rules via arrows labeled seven (7) and eight (8),…e.g., to look first for a rule for the hash value[.]" *Lambert*, [0068]. Thus, Lambert teaches *comparing* the hash value of the

software file to a set of maintained hash values, including the hash value of *the allowed program*.



Id. at Fig. 5A (annotated).

164. The combination of Dan and Lambert teaches *comparing the new program to the allowed program*. For the reasons discussed above, it would have been obvious to a person having ordinary skill in the art to configure Dan's sandbox to be able to identify and classify a new program to allow the sandbox to select the appropriate security policy even if the program does not have a certificate/RCL. *See* Claim 25(b). Based on the teachings of Lambert, it would have been obvious to a person having ordinary skill in the art to configure Dan's RCL enforcer to determine if the new application is an allowed application by comparing its hash value to a list of hashes for known applications. As recognized by Lambert, a hash "is essentially

a fingerprint that uniquely identifies a file regardless of whether the file is moved or renamed." Lambert at [0055]. Thus, a person having ordinary skill in the art would have understood that comparing a hash of the new application to the hash of the allowed application would have been a reliable way to determine if the new application is the allowed application. For these reasons, a person having ordinary skill in the art would have been motivated to combine prior art elements of Dan's RCL enforcer and Lambert's method of comparing the hash of a software file to the hash of an allowed program according to known methods to yield the predictable result of allowing Dan's RCL enforcer module to determine whether the new application is the allowed application.

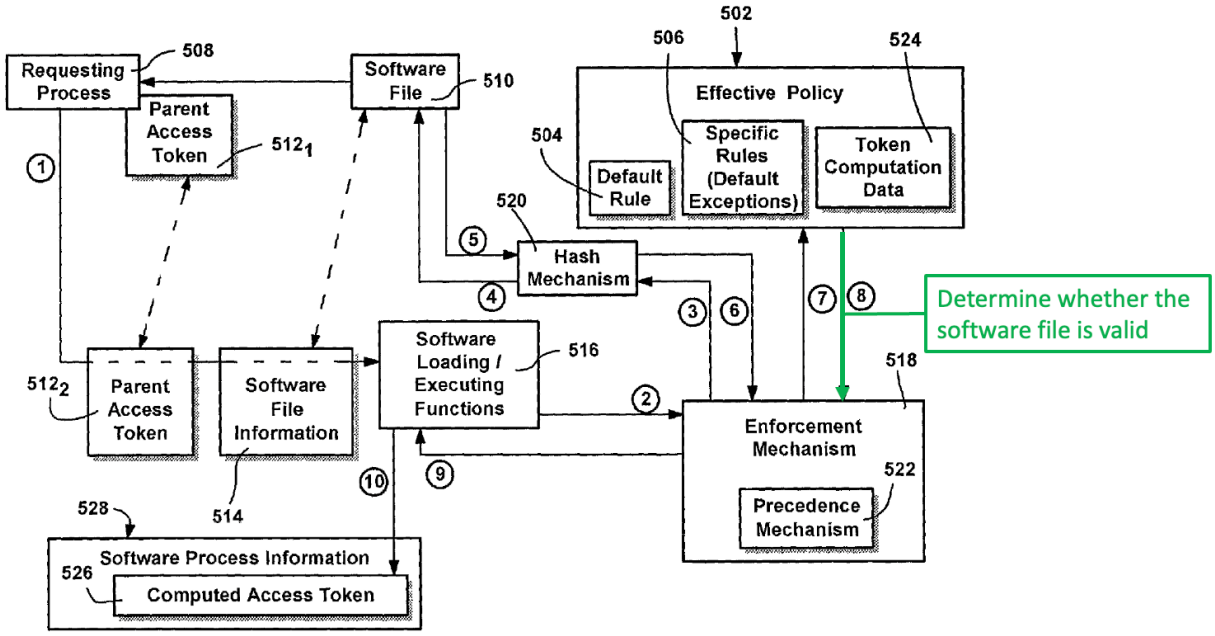
165. The '137 Patent itself acknowledges that the MD5 software modules were "commonly known to those in the art." *'137 Patent* at 8:14-15. A person having ordinary skill in the art would have had a reasonable expectation of success configuring Dan's RCL enforcer module to compute the hash of the new application using "commonly known" MD5 hash and compare it to the hash of the allowed program as taught by Lambert.

(f) *25(e) – determining whether the new program is valid;*

166. Lambert describes determining whether the software file is valid based on comparing the software file's hash value to the set of maintained hash values "to determine whether a rule exists for it." *Lambert* at [0055]. "In other words, before

opening a file for execution, a hash value is determined from that file's contents, and the set of hash value information (e.g., in the policy object) searched to determine if a rule exists for that hash value. If so, that rule provides the security level for the software." *Id.* "[I]n the case of known good code that is allowed to execute, a hash rule can specify how it will be executed[.]" *Id.* at [0056]. Per Lambert, "a suitable hash function 520 provides a unique fingerprint of the software file 510, while the effective policy 502 may have a specific rule for that hash." *Id.* at [0068]. For example, the hash rule may determine that the application is trusted and may therefore executed in an unrestricted manner. *Id.* at [0008] ("Depending on what the rule specifies for the classification that is determined for any given software module (e.g., software file), the software may execute in a normal (unrestricted) execution environment (context)…"). Alternatively, the hash rule may determine that the application is untrusted and must be executed in a restricted manner. *Id.* ("Depending on what the rule specifies for the classification that is determined for any given software module (e.g., software file), the software may execute … in a restricted execution environment…"). If there are no rules associated with the hash value, configurable default rules apply. *Id.* at [0052] ("An administrator identifies/classifies the software and sets the rules, including a default rule that applies to any software file not classified under a specific rule."). As shown below in Fig. 5A, the

enforcement mechanism 518 determines whether the software file is valid based on the hash rule returned from the effective policy 502:



Id. at Fig. 5A (annotated).

167. The combination of Dan and Lambert renders obvious *determining whether the new program is valid*. Based on the teachings of Lambert, it would have been obvious, and a person having ordinary skill in the art would have been motivated to configure Dan's RCL enforcer to determine whether the new application (even in the absence of a certificate/RCL) is valid based on a rule associated with the new application's hash value. Specifically, it would have been obvious to configure Dan's RCL enforcer to determine if the new application is valid and does not need monitoring or if the new application is not valid and must operate in a restricted environment where its access to the computing device's physical and

logical resources is monitored. A person having ordinary skill in the art would have understood that using a rule associated with the new application's hash value to determine whether the new application is valid would have been an effective and reliable way to distinguish trusted applications that do not need to be monitored from untrusted applications that do need to be monitored. Therefore, a person having ordinary skill in the art would have been motivated to combine prior art elements of Dan's RCL enforcer and Lambert's method of determining of whether a software file is valid based on a rule associated with its hash value according to known methods to yield the predictable result of allowing Dan's RCL enforcer module to determine whether the new program is trusted and does not need to be monitored during execution or whether the new program is untrusted and needs to be monitored during execution.

168. A person having ordinary skill in the art would have understood that configuring Dan's RCL enforcer to perform an initial check to determine whether the new application is valid would have represented a minor software change. Given the predictability of computer programming at the time of the '137 Patent, a person having ordinary skill in the art would have had a reasonable expectation of success configuring Dan's RCL enforcer to determine whether the new application is valid based on a rule associated with its hash value.

(g) *25(f) – if the new program is valid, permitting the new program to execute on the computing device; and*

169. *See* Claim 6(c).

(h) *25(g)(i) – if the new program is not valid, monitoring the new program while allowing it to execute on the computing device,*

170. *See* Claim 6(d)(i).

(i) *25(g)(ii) – wherein the step of monitoring the new program while allowing it to execute is performed at the operating system kernel of the computing device.*

171. *See* Claim 6(d)(ii).

G. Ground 2: The Combination of Dan, Lambert, and Schmid Renders Claims 1 and 2 Obvious

1. Claim 1

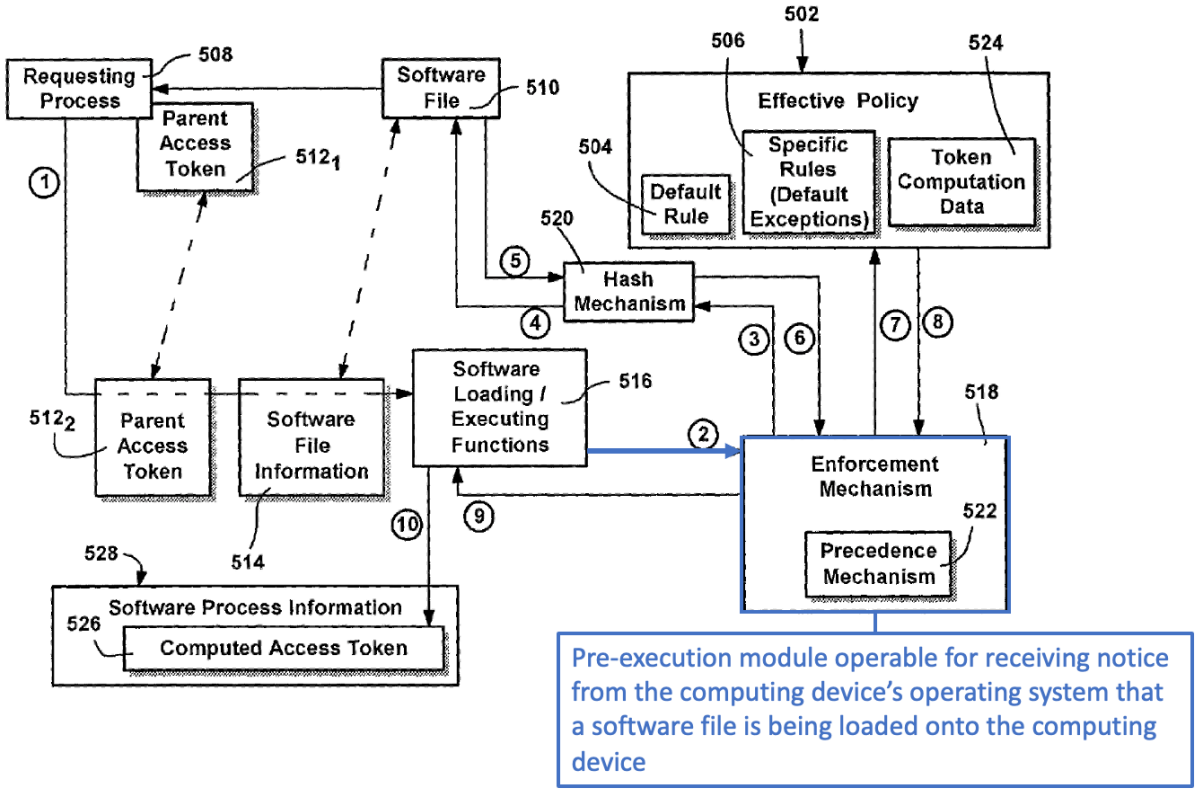
(a) *1[Pre]. – A system for managing security of a computing device comprising:*

172. Lambert describes "[a] system…that automatically, transparently and **securely controls software execution** by identifying and classifying software, and locating a rule and associated **security** level for executing executable software." *Lambert* at Abstract (emphasis added). Lambert's system operates on a "computing system environment 100," which includes "a general purpose computing device in the form of a computer 110." *Id.* at [0026], [0029], Fig. 1. Lambert describes *a system for managing security of a computing device*.

(b) *1(a) – a pre-execution module operable for receiving notice from the computing device's operating system that a new program is being loaded onto the computing device;*

173. Lambert's system incorporates an enforcement mechanism 518 that *receives notice from the computing device's operating system that* a software file 510 *is being loaded onto the computing device*. Specifically, "in FIG. 5A, a process 508 such as a process of a user or application requesting to open a software file 510 has an access token 512_1 associated with it[.]" *Lambert* at [0066]. "When the process 508 wants to **load and possibly execute the software file 510**, it identifies itself and provides information 514 identifying the software file to an appropriate function 516 (of a set) that can open, **load and/or execute** the file 510." *Id.* (emphasis added). "For example, this can be accomplished by placing an application programming interface (API) call to **the operating system**, wherein the file information 514, typically in the form a path and filename is passed, as a parameter." *Id.* (emphasis added). "Note that in a typical implementation, the **software opening function 516 comprises an operating system component**, and thus has access to a copy 512_2 of the parent access token." *Id.* (emphasis added). "Further note that **such function calls are already the typical way in which files are loaded**, and thus the process 508 (which may be of an existing application program or other type of component) need not be modified in any way." *Id.* (emphasis added). "[I]nstead of simply loading the file and executing any executable code therein with the parent token's rights and privileges, each of the functions 516 is arranged to **first provide the software file information 514 to an enforcement mechanism (circled numeral two (2) in FIG.**

5A)." *Id.* at [0067] (emphasis added). In summary, Lambert's enforcement mechanism 518 is *a pre-execution module operable for receiving notice* (i.e., software file information 514) *from the computing device's operating system* (i.e., software loading/executing functions 516) *that a* software file *is being loaded onto the computing device*:



Id. at Fig. 5A. (annotated).

174. Lambert teaches that the software file 510 is a *program*. As an example, Lambert teaches where the software file is an "e-mail program." *Id.* at [0053]. However, Lambert does not distinguish between old and new programs and encompasses both. Dan teaches a method for "Installation of Untrusted

Applications" where "[t]he application and RCL are downloaded to the host (via network, diskette, CD-ROM, satellite, etc.)." *Dan* at 20. Dan's application is *new* because it is newly downloaded to the host computer. Downloaded applications may also be *new* in the sense that they have not been executed by the client computing device before. For the reasons discussed above, Dan teaches *a new program* that is downloaded to a host computer "via network, diskette, CD-ROM, satellite, etc." *See* Claim 6(a); *Dan* at 20.

175. Based on the teachings of Dan, it would have been obvious to a person having ordinary skill in the art that Lambert's software file 510 would correspond to a newly downloaded program. Lambert acknowledges that "with the rise in the usage of networks, email and the Internet for business computing, users find themselves exposed to a wide variety of executable programs including content that is not identifiable as being executable in advance." *Lambert* at [0007]. A person having ordinary skill in the art would have understood that the computing device would receive software files via networks, email, and the Internet, and that, in this instance, the software file would represent *a new program*.

176. Moreover, a person having ordinary skill in the art would have understood that Lambert's enforcement mechanism is operable to receive notice when any software file – new or preexisting – is being loaded for execution on the computing device. For these reasons, a person having ordinary skill in the art would

have been motivated to combine prior art elements according to known methods to yield the predictable result of notifying Lambert's enforcement mechanism when a new program is loading. There would have been a reasonable expectation of success since Lambert's enforcement mechanism is notified when any software file (new or otherwise) is loading.

(c) *1(b) – a validation module coupled to the pre-execution monitor operable for determining whether the program is valid;*

177. I have been asked to interpret *the pre-execution monitor* as *the pre-execution module*. *See* Section **Error! Reference source not found.**.

178. Lambert teaches an effective policy 502 that is coupled to the enforcement mechanism 518 (i.e., *the pre-execution module*) that is operable for determining whether a software file is valid. *Lambert* at Fig. 5A. Specifically, Lambert teaches that after the enforcement mechanism 518 receives the notification from the software loading/executing functions 516, it "call[s] (or include in its own code) a hash function 520 or the like to obtain a hash value from the contents of the software file 510, as generally represented in FIG. 5A by the arrows labeled with circled numerals three (3) to six (6)." *Id.* at [0068]. "[A] suitable hash function 520 provides a unique fingerprint of the software file 510, while the effective policy 502 may have a specific rule for that hash." *Id*. For example, Lambert teaches that the

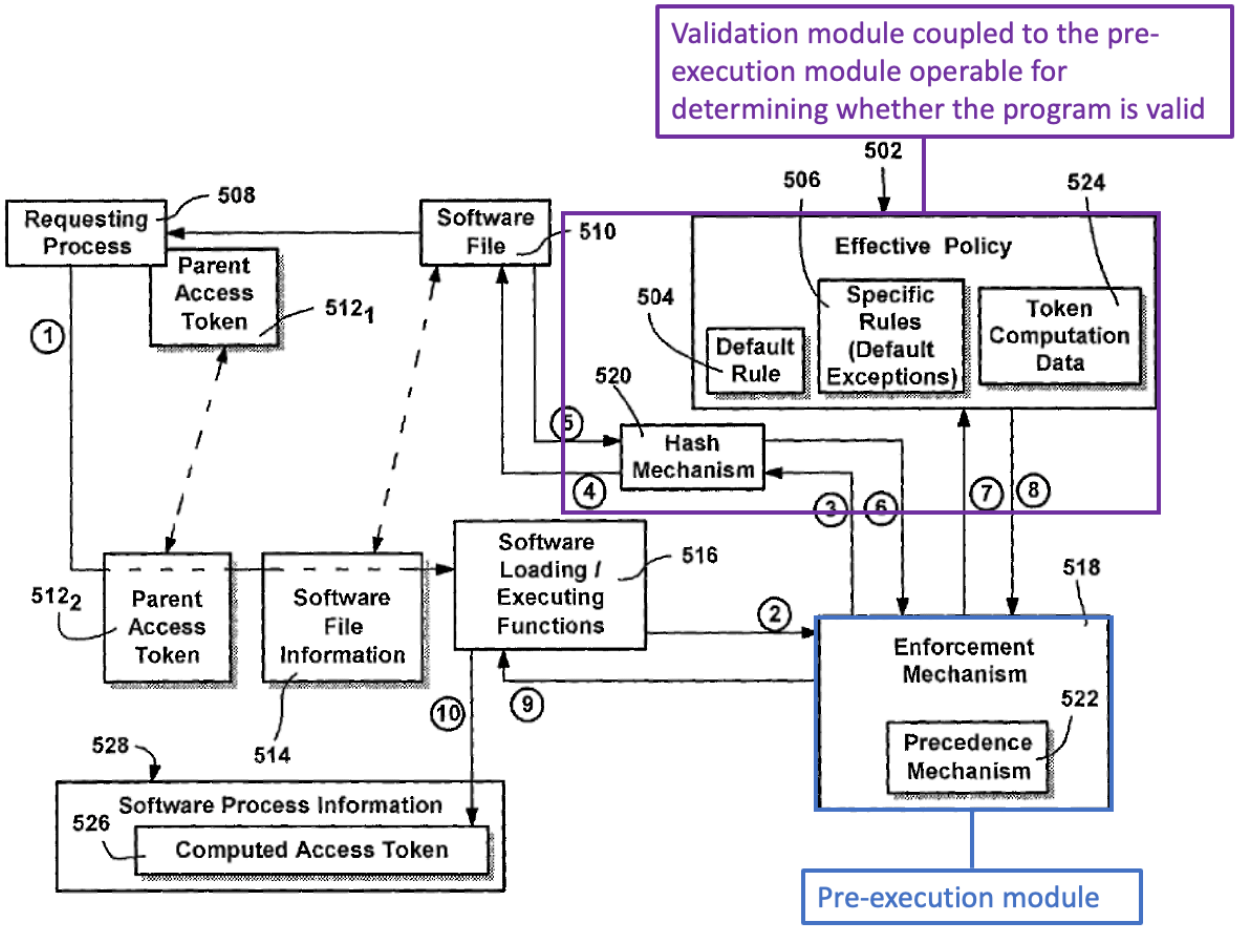
hash is "a unique hash value of the file's content (e.g., via an MD5 image hash)." *Id.* at [0053].

179. To determine if the software file's hash matches the hash of a known software file, "the set of maintained hash values can be queried to determine whether a rule exists for it." *Id.* at [0055]. "In other words, before opening a file for execution, a hash value is determined from that file's contents, and the set of hash value information (e.g., in the policy object) [is] searched to determine if a rule exists for that hash value." *Id.* The effective policy 502 includes the set of hash value information for known software files and their associated rules. Specifically, "the effective policy 502 generally comprises the general (default) rule 504 and a set of specific rules 506 which set forth the exceptions to the default rule 502." *Id.* at [0065]. "Note that while FIG. 5A essentially represents accessing the policy to get the rule or rules via arrows labeled seven (7) and eight (8),…e.g., to look first for a rule for the hash value[.]" *Id.* at [0068].

180. The rules stored in the policy 502 dictate whether the software file is valid. Specifically, "Via a user interface or the like operably connected to a suitable hash mechanism, an administrator can select any software file, obtain a hash value from its contents, and enter and associate a rule (that specifies a security level) with that hash value." *Id.* at [0055]. "Multiple files can have their corresponding hash values and rules maintained in **a policy object**…whereby for any given instance of

any file, the set of maintained hash values can be queried to determine whether a rule exists for it." *Id.* "In this manner, an administrator can set a rule for any file based on its unique hash value, with the rule specifying whether (and if so, how) a software file having that hash value will execute." *Id.* at [0056]. "In the case of known bad code, the administrator may set the rule to specify that any file with that hash will not be allowed to execute at all (i.e., 'disallowed' or 'revoked' status)." *Id.* "[I]n in the case of known good code that is allowed to execute, a hash rule can specify how it will be executed, e.g., unrestricted (normal) or restricted (and to what extent)." *Id.*

181. Thus, Lambert's effective policy 502 and hash mechanism 520 collectively qualify as *a validation module coupled to the pre-execution monitor operable for determining whether the program is valid*:



Id. at Fig. 5A (annotated).

(d) *1(c) – a detection module coupled to the pre-execution monitor operable for intercepting a trigger from the computing device's operating system;*

182. I have been asked to interpret *the pre-execution monitor* as *the pre-execution module*. *See* Section **Error! Reference source not found.**.

183. Lambert teaches an instance where the application is allowed to execute in a restricted environment. *Lambert* at [0008] ("Depending on what the rule specifies for the classification that is determined for any given software module (e.g., software file), the software may execute … in a restricted execution

environment…"). Per Lambert, a restricted execution environment "reduces software's access to resources, and/or removes privileges, relative to a user's normal access token." *Id.* at Abstract. When it is determined that software should be restricted, "the enforcement mechanism 518 can compute and associate a restricted token with the software file 510, whereby if creation of a process is requested for the software file, (it includes executable content), any process of the software file is restricted in its access rights and privileges according to the restricted token." *Id.* at [0070]. When the user requests to load a software file, "the enforcement mechanism 518 completes the call and returns information back to the software function 516 that called it, the returned information essentially including an instruction on whether to open the file, and if so, the token to associate with the file." *Id.* at [0071]. "If the computed token 526 is a restricted token, the process of the software file can now execute, but only in association with the restricted token, thus providing the rule-determined secure execution environment." *Id.*

184. In Figure 3, an untrusted process 300 having an associated restricted token 210 attempts to access object 302. *Id.* at Fig. 3, [0042]. "When the process 300 desires access to an object 302, the process 300 specifies the type of access it desires (e.g., obtain read/write access to a file object), and the operating system (e.g., kernel) level provides its associated token to an object manager 304." *Id.* Thus, Lambert's

system presumes that an untrusted process will notify the object manager 304 when attempting to access a file object 302.

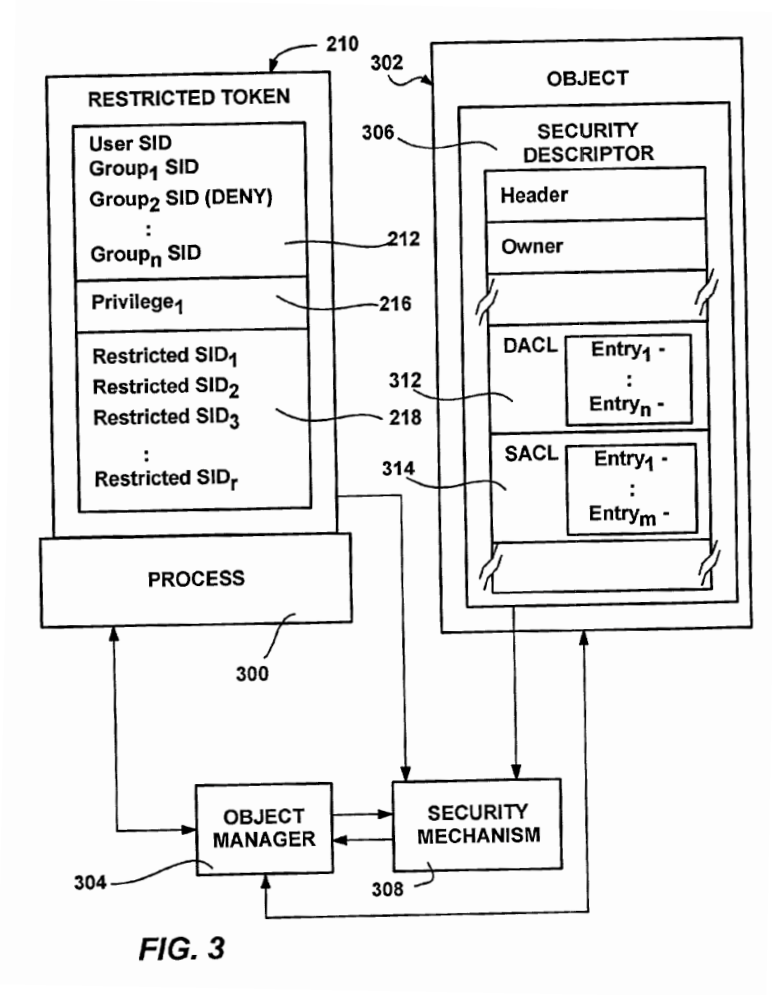


FIG. 3

185.

Id. at Fig. 3.

186. As I discussed above, Lambert describes a two-phase system including a pre-execution phase, which is represented by Figure 5A and its corresponding disclosure, and an execution phase, which is represented by Figure 3 and its corresponding disclosure, which describes how the system operates during restricted program execution. *See* Section III.E.2. Thus, a person having ordinary skill in the

art would understand that the disclosures of Figures 3 and 5A are clearly related and do not represent different embodiments of Lambert.

187. A person having ordinary skill in the art would have understood that an untrusted process may attempt to bypass the object manager and directly access the operating system kernel to access the file object 302. For example, Schmid recognizes that "While most well-behaved Windows applications make system calls through the Win32 API, it is also possible to bypass this interface by making calls directly to kernel services." *Schmid* at 5:29-32. "System services are operations performed by an operating system kernel on behalf of applications or other kernel components." *Id.* at 5:49-51. "For instance, manipulating CPU hardware, starting and stopping processes, and manipulating files are sensitive operations generally implemented by services at the kernel level." *Id.* at 5:53-56. Given Lambert's system is also implemented in a Windows environment, a person having ordinary skill in the art would have understood that an untrusted and potentially malicious software process may bypass Lambert's object manager 304 and access the file object 302 directly via the Windows operating system kernel. *Lambert* at [0037] ("One way that this can be accomplished with an operating system (e.g., Microsoft Corporation's Windows® 2000)…").

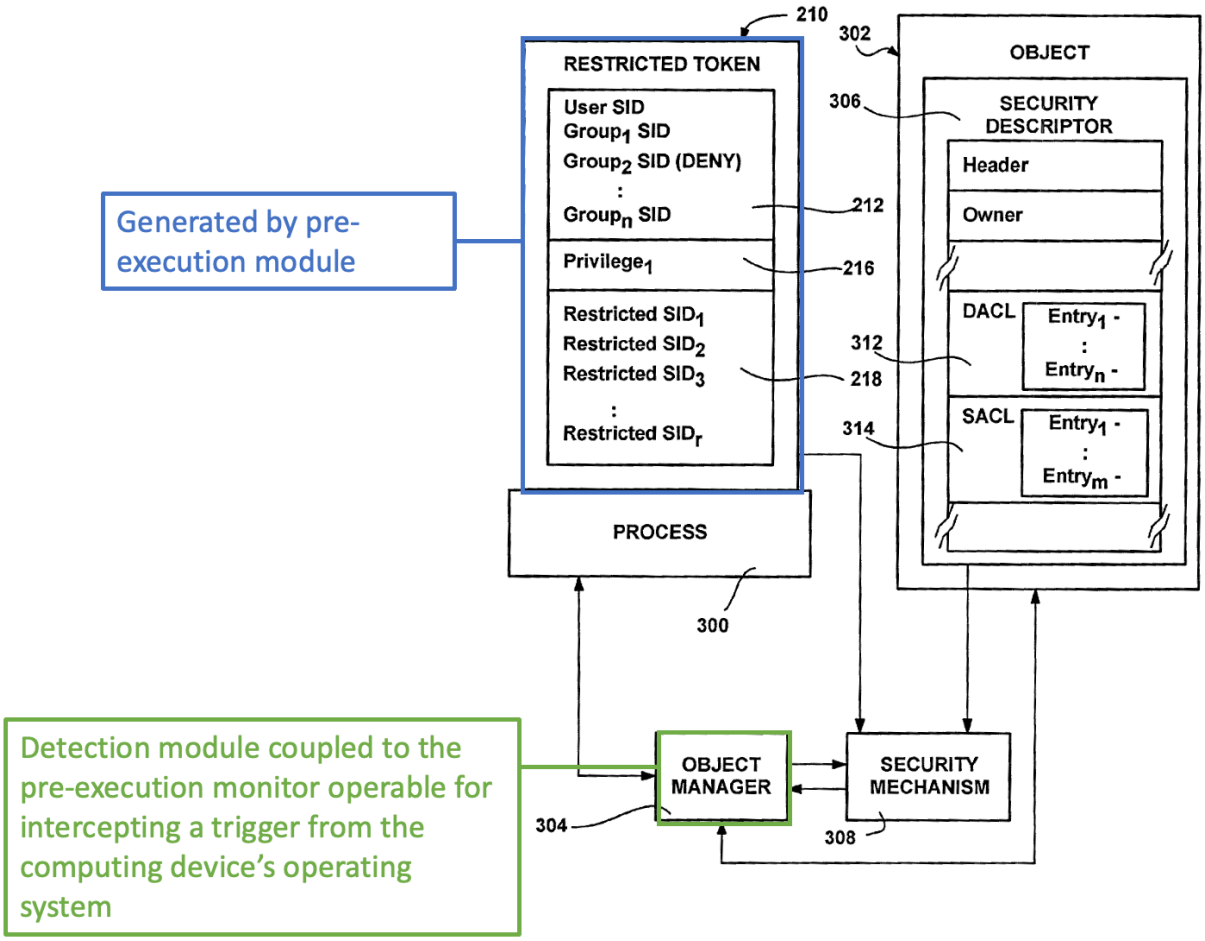
188. To prevent an untrusted application from bypassing user mode protections, Schmid teaches a device driver that is used "[t]o implement a non-

bypassable kernel wrapper" that "intercept[s] system service requests." *Schmid* at 5:40-48. Schmid's device driver intercepts triggers from the computer's operating system. Specifically, Schmid teaches that in a Windows environment, "user applications invoke system services by executing an interrupt instruction." *Id.* at 5:57-59. "A kernel entity, known as a dispatcher, initially responds to an interrupt instruction, determines the nature of an interrupt, and calls a function to handle the request." *Id.* at 5:62-64. "Two tables in kernel memory describe locations and parameter requirements of all functions available to a dispatcher," including a table that "specifies handlers for user requests." *Id.* at 5:63-65. Schmid teaches "constructing a device driver that may be dynamically loaded into a kernel, or may be loaded as part of a boot sequence." *Id.* at 6:6-9. "When a device driver loads, it modifies an entry in a table consulted by a dispatcher to handle an interrupt instruction." *Id.* at 6:9-11. "[T]he modified entry may cause a dispatcher to call an inserted function" instead of the intended operating system function. *Id.* at 6:11-15. Schmid further teaches "that the driver modifies only tables relevant to requests from user applications." *Id.* at 6:15-16. Thus, a person having ordinary skill in the art would understand that Schmid's device driver intercepts a trigger from the operating system's dispatcher by modifying entries in the table consulted by the dispatcher to handle an interrupt instruction. As such, Schmid's device driver qualifies as *a*

detection module operable for intercepting a trigger from the computing device's operating system.

189. Lambert only provides a functional description of the object manager 304 and does not provide details as to how it is to be implemented. For example, Lambert does not describe whether the object manager is implemented in user mode or kernel mode or how the object manager is notified when the untrusted program attempts to access an object 302. Based on the teachings of Schmid, it would have been obvious, and a person having ordinary skill in the art would have been motivated to implement Lambert's object manager using Schmid's device driver (i.e., *a detection module*). Specifically, it would have been obvious to configure Lambert's system to implement the object manager by loading a device driver that modifies entries in a table consulted by the operating system's dispatcher and inserts corresponding functions that access the untrusted process's restricted token and the object 302's security descriptor. *Lambert* at [0042]-[0043]. A person having ordinary skill in the art would have understood that this would have ensured that Lambert's object manager is notified when the untrusted process attempts to access an object 302 by making calls either via the Win32 API or via the operating system kernel. Since the object manager accesses the untrusted process's restricted token and the restricted token is generated by the enforcement mechanism (i.e., *the pre-execution module*), the object manager is coupled to the enforcement mechanism.

Thus, the combination of Lambert and Schmid teaches *a detection module coupled to the pre-execution monitor operable for intercepting a trigger from the computing device's operating system* as illustrated in the following annotated figure:



Lambert at Fig. 3A (annotated).

190. Given the above-described advantages of implementing Lambert's object manager such that it cannot be bypassed by a malicious program, a person having ordinary skill in the art would have been motivated to implement Lambert's object manager using Schmid's device driver to improve similar computer security

systems in the same way. Indeed, this modification would have ensured that Lambert's object manager would be aware when an untrusted program is attempting to access a physical or logical object that it may be restricted from accessing via its restricted token. Thus, this modification would have predictably enhanced the robustness of Lambert's computer security system by ensuring that a malicious program is not able to bypass its protections by making calls directly to the operating system kernel.

191. Writing a device driver, such as described by Schmid, was a well-known and predictable way to inject custom code into the kernel space of a Windows computing device. *See* Section III.C.1. Likewise, the operation of the Windows operating system, including the operation of the dispatcher, was well documented and understood by a person having ordinary skill in the art. For these reasons, there would have been a reasonable expectation of success implementing Lambert's object manager using Schmid's device driver.

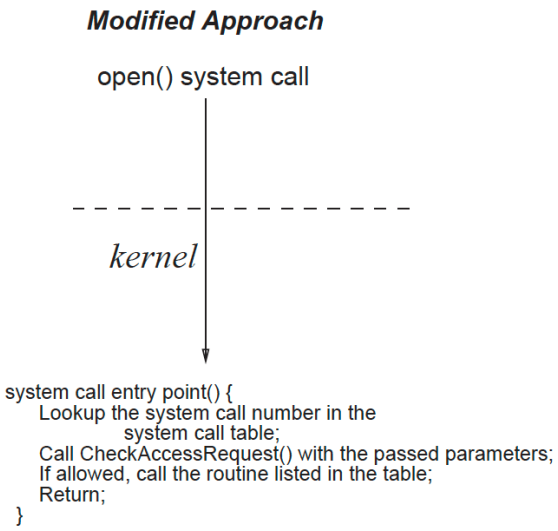
(e) *1(d) – and an execution module coupled to the detection module and operable for monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module.*

192. Lambert teaches a security mechanism 308 that receives the untrusted process's restricted token 210 and object 302's security descriptor 306 from the object manager 304 (i.e., *the detection module*). *Lambert* at [0043] ("The object 302

has a security descriptor 310 associated therewith, and the object manager 304 provides the security descriptor 306 and the restricted token 210 to a security mechanism 308."). "The security mechanism 308 compares the security IDs in the restricted token 210 along with the type of action or actions requested by the process 300 against the entries in the DACL [discretionary access control list] 312," where the DACL includes "access control entries, and for each entry, one or more access rights (allowed or denied actions) corresponding to that entry." *Id.* at [0043]. "If a match is found with an allowed user or group, and the type of access desired is allowable for the user or group, a handle to the object 302 is returned to the process 300, otherwise access is denied." *Id.* Again, Lambert provides a functional description of the security mechanism but does not provide implementation details, such as whether the security mechanism is implemented in user mode or kernel mode.

193. Dan teaches a similar monitoring system, including an RCL enforcer module that "ensures that the permissions and resource consumption limits specified in the RCL are not violated." *Dan* at 9. To accomplish this, Dan teaches that the "RCL enforcer monitors all systems calls, i.e., access to requests to logical resources in the system." *Id.* at 12. "This monitoring requires the RCL enforcer to be invoked for each system call made by an application." *Id.* "The RCL enforcer then validates the parameters of the system call against those specified in the RLRT data structures

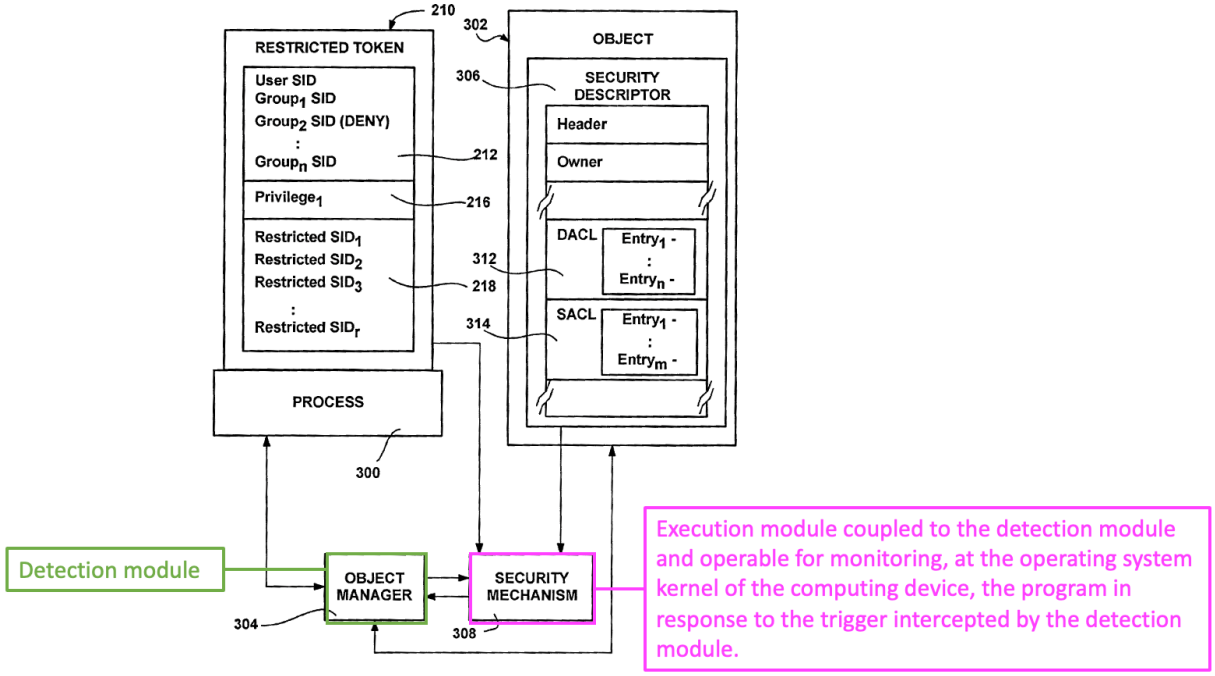
and either allows or disallows the call from continuing." *Id.* Unlike Lambert, Dan does provide implementation details for the RCL enforcer. Specifically, Dan teaches a system call modification approach where "the operating system's call interfaces is modified to call the `CheckAccessRequest()` upon entry to verify the parameters of the system call." *Id.* at 14. As shown in the figure below, this monitoring occurs on the operating system kernel:



Id. at 15. Per Dan, this approach is preferred "because of the consistency it offers for enforcing security" and it "will ensure consistent security checks for all applications that are executed on the machine." *Id.* at 14-15.

194. As described above, the combination of Lambert and Schmid teaches that the object manager 304 is implemented via a device driver loaded into the operating system kernel. *See* Claim 1(c). Since the security mechanism works in tandem with the object manager to monitor the untrusted process's behavior, it

would also been obvious to a person having ordinary skill in the art to also implement the security mechanism at the computing device's operating system kernel level as taught by Dan. Implementing the security mechanism in the kernel would have allowed it to monitor the operating system calls intercepted by the object manager and ensure that the untrusted process is in compliance with its associated restricted token. Thus, the combination of Lambert, Schmid, and Dan teaches *an execution module coupled to the detection module and operable for monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module* as illustrated in the following annotated figure:



Lambert at Fig. 3 (annotated).

195. I have been asked to interpret this limitation as means-plus-function where the function is "monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module" and the corresponding structure is "if the program was not validated, then monitor the nonvalidated program in response to triggers while the program is executing." *See* Section **Error! Reference source not found.** For the reasons discussed above, the Lambert-Schmid-Dan combination teaches a security mechanism 308 that monitors an untrusted (i.e., not validated) program in response to triggers intercepted by the object manager while the untrusted program is executing. Additionally, this structure performs the function of "monitoring, at the operating system kernel of the computing device, the program in response to the trigger intercepted by the detection module" for the same reasons.

196. A person having ordinary skill in the art would have understood that implementing Lambert's security mechanism on the kernel level, as taught by Dan, would have allowed the object manager, which is also operating in the computing device's kernel space, to efficiently communicate with the security mechanism without requiring communication between kernel mode and user mode. Moreover, a person having ordinary skill in the art would have understood that maintaining the security mechanism in the kernel mode as opposed to the user mode would have decreased the probability of a malicious program or user attempting to disable or

tamper with the security mechanism. As taught by Dan, a person of ordinary skill would have further recognized that implementing Lambert's security mechanism in the kernel would have offered increased consistency for enforcing security. For these reasons, a person having ordinary skill in the art would have been motivated to combine prior art elements according to known methods and implement Lambert's security mechanism in the operating system kernel so that it can monitor the program in response to triggers intercepted by the object manager. Due to the increased security, there would have been a reasonable expectation of success implementing Lambert's security mechanism in the operating system kernel.

2. Claim 2 - The system of claim 1, wherein the pre-execution module is further operable for suspending loading of the program onto the computing device.

197. As I discussed above in Claim 25(c), Lambert's enforcement mechanism 518 (i.e., *the pre-execution module*) is suspends the loading of the program onto the computing device. *See* Claim 25(c).

IV. CONCLUSION

198. I declare that all statements made herein of my knowledge are true, and that all statements made on information and belief are believed to be true, and that these statements were made with the 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.

Date: October 24, 2023

By: _____
Dr. Markus Jakobsson

CV and Research Statement

Markus Jakobsson
www.linkedin.com/in/markusjakobsson
www.markus-jakobsson.com

1 At a Glance

- **Focus.** *Identification of security problems, trends and solution along four axes* – computational, structural, physical *and* social*; quantitative and qualitative fraud analysis; development of disruptive security technologies.*
- **Education.** *PhD* (Computer Science/Cryptography, University of California at San Diego, 1997); *MSc* (Computer Science, University of California at San Diego, 1994); *MSc* (Computer Engineering, Lund Institute of Technology, Sweden, 1993).
- **Large research labs.** *San Diego Supercomputer Center* (Researcher, 1996-1997); *Bell Labs* (Member of Technical Staff, 1997-2001); *RSA Labs* (Principal Research Scientist, 2001-2004); *Xerox PARC* (Principal Scientist, 2008-2010); PayPal (Principal Scientist of Consumer Security, Director, 2010-2013); Qualcomm (Senior Director, 2013-2015); Agari (Chief Scientist, 2016–2018); Amber Solutions Inc (Chief of Security and Data Analytics, 2018 – 2019); ByteDance (Principal Scientist, 2020-2021)
- **Academia.** *New York University* (Adjunct Associate Professor, 2002-2004); *Indiana University* (Associate Professor & Associate Director, 2004-2008; Adjunct Associate Professor, 2008-2016).
- **Entrepreneurial activity.** *ZapFraud* (Anti-scam technology; CTO and founder, 2012-current); *RavenWhite Security* (Authentication solutions; CTO and founder, 2005-); *RightQuestion* (Consulting; Founder, 2007-current); *FatSkunk* (Malware detection; CTO and founder, 2009-2013 – FatSkunk was acquired by Qualcomm); *LifeLock* (Id theft protection; Member of fraud advisory board, 2009-2013); *CellFony* (Mobile security; Member of technical advisory board, 2009-2013); *PopGiro* (User Reputation; Member of technical advisory board, 2012-2013); *MobiSocial* (Social networking, Member of technical advisory board, 2013); *Cequence Security* (Anti-fraud, Member of technical advisory board, 2013–current)
- **Anti-fraud consulting.** *KommuneData* [Danish govt. entity] (1996); *J.P. Morgan Chase* (2006-2007); *PayPal* (2007-2011); *Boku* (2009-2010); *Western Union* (2009-2010).

- **Intellectual Property, Testifying Expert Witness.** *Inventor of 100+ patents; expert witness in 25+ patent litigation cases* (McDermott, Will & Emery; Bereskin & Parr; WilmerHale; Hunton & Williams; Quinn Emanuel Urquhart & Sullivan; Freed & Weiss; Berry & Domer; Fish & Richardson; DLA Piper; Cipher Law Group; Keker & Van Nest). Details and references upon request.

- **Publications**. Books: *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft* (Wiley, 2006); *Crimeware: Understanding New Attacks and Defenses* (Symantec Press, 2008); *Towards Trustworthy Elections: New Directions in Electronic Voting* (Springer Verlag, 2010); *Mobile Authentication: Problems and Solutions*)Springer Verlag, 2012); *The Death of the Internet* (Wiley, 2012); *Understanding Social Engineering* (Springer Verlag, 2016); *Security, Privacy and User Interaction* (Springer Verlag, 2020); *100+ peer-reviewed publications*

2 At a Glance

Ten years before Bitcoin was created, I formalized the notion of Proof of Work and described its use for mining of crypto payments. I later developed energy-efficient alternatives to this paradigm, and showed how to enable mining on mobile devices, which is not possible for Bitcoin. I am the founder of the academic discipline of phishing and have developed techniques to predict fraud trends years before they emerge, enabling countermeasures to be developed before they are needed. I developed the notion of implicit authentication, which is now ubiquitous; I also founded a company that developed the first retroactive virus detection technology, with guarantees of detection; the company was acquired by Qualcomm in 2013. I have worked as chief scientist and similar positions in startups as well as industry behemoths, such as PayPal. I have several hundred patents to my name and am a prominent security researcher with hundreds of peer reviewed publications and an array of textbooks. My 1997 PhD thesis, from University of California at San Diego, was on distributed electronic payment systems with revocable privacy.

3 Work History (Highlights)

1. **Member of Technical Staff, Bell Labs (1997-2001).** Markus was part of the security research group at Bell Labs. He formalized the notion of *proof of work*, later an integral part of BitCoin.

2. **Principal Scientist, RSA Labs (2001-2005).** Markus posited that phishing would become a mainstream problem, and developed ethical techniques for identifying likely trends based on human subject experiments.

3. **Associate Professor, Indiana University (2005-2008).** Markus was hired to lead the newly formed security group at Indiana University, and

created a research group comprising approximately 10 professors and 30 students, studying social engineering and fraud.

4. **Principal Scientist, Xerox PARC (2008-2010).** Markus was hired to lead the research efforts of the Xerox PARC security group, and developed the notion of *implicit authentication*, a technology that is now ubiquitous.

5. **Principal Scientist, PayPal (2010-2013).** Markus did research on security and user interfaces, and developed techniques to reduce the losses associated with *liar buyer fraud*.

6. **Senior Director, Qualcomm (2013-2016).** Qualcomm acquires FatSkunk, a company founded by Markus. At FatSkunk, Markus developed *retroactive* malware detection with provable security guarantees. A simplified version of this is now deployed with almost all Qualcomm chipsets.

7. **Chief Scientist, Agari (2016-2018).** Markus developed a technique to acquire fraudster intelligence by compromising scammer email accounts – *while staying within the law* – resulting in the extradition of several African scam lords to the U.S.

8. **Chief of Security and Data Analytics, Amber Solutions (2018-2020).** Markus developed usable configuration methods supporting improved security and privacy for IoT installations.

9. **Chief Scientist, ByteDance (2020-2021).** Markus oversaw the establishment of a research group and a research agenda at ByteDance, and contributed to their intellectual property and product security.

10. **Chief Scientist, Artema LABS (2021-).** Managing the research, product strategy, and patent strategy for Artema LABS, a Los Angeles based startup in the Crypto/NFT sector.

4 Publication List

Books (1-8); book chapters, journals, conference publications and other scientific publications (9-147), issued /published U.S. patents (148-234). For an updated list, and for international patents, please see `www.markus-jakobsson.com/publications` and appropriate patent search engines.

References

[1] M. Jakobsson, *Security, Privacy and User Interaction*, ISBN 978-3-030-43753-4, 110 pages, 2020.

[2] M. Jakobsson, *Understanding Social Engineering Based Scams*, ISBN 978-1-4939-6457-4, 130 pages, Springer, 2016.

[3] M. Jakobsson, *Mobile Authentication: Problems and Solutions*, ISBN 1461448778, 125 pages, Springer, 2013.

[4] M. Jakobsson, (editor) *The Death of the Internet*, ASIN B009CN2JVE, 359 pages, IEEE Computer Society Press, 2012.

[5] D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. Ryan, J. Benaloh, and M. Kutylowski, (editors), *Towards Trustworthy Elections: New Directions in Electronic Voting*, 411 pages, (Vol. 6000), Springer, 2010.

[6] M. Jakobsson and Z. Ramzan (editors), *Crimeware: Trends in Attacks and Countermeasures*, ISBN 0321501950, Hardcover, 582 pages, Symantec Press / Addison Wesley, 2008.

[7] M. Jakobsson and S. A. Myers (editors), *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*, ISBN 0-471-78245-9, Hardcover, 739 pages, Wiley, 2006.

[8] M. Jakobsson, M. Yung, J. Zhou, *Applied Cryptography and Network Security: Second International Conference , Yellow Mountain, China, 2004*, 511 pages, Lecture Notes in Computer Science (Book 3089), 2004.

[9] M. Jakobsson, "Permissions and Privacy," in IEEE Security & Privacy, vol. 18, no. 2, pp. 46-55, March-April 2020

[10] M. Jakobsson, "The Rising Threat of Launchpad Attacks," in IEEE Security & Privacy, vol. 17, no. 5, pp. 68-72, Sept.-Oct. 2019

[11] J Koven, C Felix, H Siadati, M Jakobsson, E Bertini, "Lessons learned developing a visual analytics solution for investigative analysis of scamming activities," IEEE transactions on visualization and computer graphics 25 (1), 225-234

[12] M Jakobsson, "Two-factor inauthentication?the rise in SMS phishing attacks" Computer Fraud & Security 2018 (6), 6-8

[13] M. Dhiman, M. Jakobsson, T.-F. Yen, "Breaking and fixing content-based filtering," 2017 APWG Symposium on Electronic Crime Research (eCrime), 52-56

[14] M. Jakobsson, "Addressing sophisticated email attacks," 2017 International Conference on Financial Cryptography and Data Security, 310-317

[15] M. Jakobsson, "User trust assessment: a new approach to combat deception," Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust, 2016, pages 73-78

[16] H. Siadati, T. Nguyen, P. Gupta, M. Jakobsson, "Mind your SMSes: Mitigating social engineering in second factor authentication," Computers and Security, 2016

[17] M. Jakobsson, W. Leddy, "Could you fall for a scam? Spam filters are passe. What we need is software that unmasks fraudsters," IEEE Spectrum 53 (5), 2016, 40-55

[18] N. Sae-Bae, M. Jakobsson, *Hand Authentication on Multi-Touch Tablets*, HotMobile 2014

[19] Y. Park, J. Jones, D. McCoy, E. Shi, M. Jakobsson, *Scambaiter: Understanding Targeted Nigerian Scams on Craigslist*, NDSS 2014

[20] D. Balfanz, R. Chow, O. Eisen, M. Jakobsson, S. Kirsch, S. Matsumoto, J. Molina, and P. van Oorschot, "The future of authentication," Security & Privacy, IEEE, 10(1), 22-27, 2012.

[21] M. Jakobsson, and H. Siadati, *Improved Visual Preference Authentication: Socio-Technical Aspects in Security and Trust*, (STAST), 2012 Workshop on IEEE, 27–34, 2012.

[22] M. Jakobsson, R. I. Chow, and J. Molina, "Authentication-Are We Doing Well Enough?[Guest Editors' Introduction]" Security & Privacy, IEEE, 10(1), 19-21, 2012.

[23] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, "Implicit authentication through learning user behavior," Information Security, 99-113, Springer Berlin Heidelberg, 2011.

[24] M. Jakobsson and K. Johansson, "Practical and Secure Software-Based Attestation," Lightweight Security & Privacy: Devices, Protocols and Applications (LightSec), 1–9, 2011.

[25] A. Juels, D. Catalano, and M.Jakobsson, *Coercion-resistant electronic elections: Towards Trustworthy Elections*, 37–63, Springer Berlin Heidelberg, 2010.

[26] M. Jakobsson and F. Menczer, "Web Forms and Untraceable DDoS Attacks," in Network Security, Huang, S., MacCallum, D., and Du, D. Z., Eds.,77–95, Springer, 2010.

[27] R. Chow, M. Jakobsson, R. Masuoka, J. Molina, Y. Niu, E. Shi, and Z. Song, "Authentication in the Clouds: A Framework and its Application to Mobile Users," 2010.

[28] X. Wang, P. Golle, M. Jakobsson, and A. Tsow, "Deterring voluntary trace disclosure in re-encryption mix-networks," ACM Trans. Inf. Syst. Secur., 13(2), 1-24, 2010.

[29] X. Wang, P. Golle, M. Jakobsson, A.Tsow, "Deterring voluntary trace disclosure in re-encryption mix-networks," ACM Trans. Inf. Syst. Secur. 13(2): (2010)

[30] M. Jakobsson, and C. Soghoian, "Social Engineering in Phishing," Information Assurance, Security and Privacy Services, 4, 2009.

[31] M. Jakobsson, C. Soghoian and S. Stamm, "Phishing," Handbook of Financial Cryptography (CRC press, 2008)

[32] M. Jakobsson and A. Tsow, "Identity Theft," In John R. Vacca, Editor, "Computer And Information Security Handbook" (Morgan Kaufmann, 2008)

[33] S. Srikwan and M. Jakobsson, "Using Cartoons to Teach Internet Security," Cryptologia, vol. 32, no. 2, 2008

[34] M. Jakobsson, N. Johnson and P. Finn, "Why and How to Perform Fraud Experiments," IEEE Security and Privacy, March/April 2008 (Vol. 6, No. 2) pp. 66-68

[35] M. Jakobsson and S. Myers, "Delayed Password Disclosure," International Journal of Applied Cryptography, 2008, pp. 47-59.

[36] M. Jakobsson and S. Stamm, "Web Camouflage: Protecting Your Clients from Browser Sniffing Attacks," IEEE Security & Privacy Magazine. November/December 2007

[37] P. Finn and M. Jakobsson, "Designing and Conducting Phishing Experiments," IEEE Technology and Society Magazine, Special Issue on Usability and Security

[38] T. Jagatic, N. Johnson, M. Jakobsson and F. Menczer. "Social Phishing," The Communications of the ACM, October 2007

[39] A. Tsow, M. Jakobsson, L. Yang and S. Wetzel, "Warkitting: the Drive-by Subversion of Wireless Home Routers," Anti-Phishing and Online Fraud, Part II Journal of Digital Forensic Practice, Volume 1, Special Issue 3, November 2006

[40] M. Gandhi, M. Jakobsson and J. Ratkiewicz, "Badvertisements: Stealthy Click-Fraud with Unwitting Accessories," Anti-Phishing and Online Fraud, Part I Journal of Digital Forensic Practice, Volume 1, Special Issue 2, November 2006

[41] N. Ben Salem, J.-P. Hubaux and M. Jakobsson. "Reputation-based Wi-Fi Deployment," Mobile Computing and Communications Review, Volume 9, Number 3 (Best papers of WMASH 2004)

[42] N. Ben Salem, J. P. Hubaux, and M. Jakobsson. "Node Cooperation in Hybrid Ad hoc Networks," IEEE Transactions on Mobile Computing, Vol. 5, No. 4, April 2006.

[43] P. MacKenzie, T. Shrimpton, and M. Jakobsson. "Threshold Password-Authenticated Key Exchange," Journal of Cryptology, 2005

[44] A. Juels, M. Jakobsson, E. Shriver, and B. Hillyer. "How To Turn Loaded Dice Into Fair Coins." IEEE Transactions on Information Theory, vol. 46(3). May 2000. pp. 911–921.

[45] M. Jakobsson, P. MacKenzie, and J.P. Stern. "Secure and Lightweight Advertising on the Web," Journal of Computer Networks, vol. 31, issue 11–16, Elsevier North-Holland, Inc., 1999. pp. 1101–1109.

[46] M. Jakobsson, "Cryptographic Protocols," Chapter from The Handbook of Information Security. Hossein Bidgoli, Editor-in-Chief. Copyright John Wiley & Sons, Inc., 2005, Hoboken, N.J.

[47] M. Jakobsson, "Cryptographic Privacy Protection Techniques," Chapter from The Handbook of Information Security. Hossein Bidgoli, Editor-in-Chief. Copyright John Wiley & Sons, Inc., 2005, Hoboken, N.J.

[48] M. Jakobsson, E. Shi, P. Golle, R. Chow, "Implicit authentication for mobile devices," 4th USENIX Workshop on Hot Topics in Security (HotSec '09); 2009 August 11; Montreal, Canada.

[49] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," Proceedings of the 2009 ACM Workshop on Cloud Computing Security (CCSW 2009); 2009 November 13; Chicago, IL. NY: ACM; 2009; pp. 85–90.

[50] M. Jakobsson, H. Siadati, M. Dhiman, "Liar Buyer Fraud, and How to Curb It," NDSS, 2015

[51] M. Jakobsson, T.-F. Yen, "How Vulnerable Are We To Scams?," BlackHat, 2015

[52] M. Jakobsson, "How to Wear Your Password," BlackHat, 2014

[53] M. Jakobsson and G. Stewart, "Mobile Malware: Why the Traditional AV Paradigm is Doomed, and How to Use Physics to Detect Undesirable Routines," in BlackHat, 2013.

[54] M. Jakobsson, and H. Siadati,"SpoofKiller: You Can Teach People How to Pay, but Not How to Pay Attention" in Socio-Technical Aspects in Security and Trust (STAST), 2012 Workshop on, 3-10, 2012.

[55] M. Jakobsson, and M. Dhiman,"The benefits of understanding passwords," in Proceedings of the 7th USENIX conference on Hot Topics in Security, Berkeley, CA, USA, 2012.

[56] M. Jakobsson, and S. Taveau, "The Case for Replacing Passwords with Biometrics," Mobile Security Technologies, 2012.

[57] M. Jakobsson and D. Liu, "Bootstrapping mobile PINs using passwords," W2SP, 2011.

[58] M. Jakobsson and R. Akavipat, "Rethinking passwords to adapt to constrained keyboards," 2011.

[59] Y. Niu, E. Shi, R. Chow, P. Golle, and M. Jakobsson, "One Experience Collecting Sensitive Mobile Data," In USER Workshop of SOUPS, 2010.

[60] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, "Implicit Authentication through Learning User Behavior," 2010.

[61] M. Jakobsson and K. Johansson, Assured Detection of Malware With Applications to Mobile Platforms, 2010.

[62] M. Jakobsson and K. Johansson, "Retroactive Detection of Malware With Applications to Mobile Platforms," in HotSec 2010, Washington, DC, 2010.

[63] M. Jakobsson, A Central Nervous System for Automatically Detecting Malware, 2009.

[64] R. Chow, P. Golle, M. Jakobsson, R. Masuoka, J. Molina, E. Shi, and J. Staddon, "Controlling data in the cloud: outsourcing computation without outsourcing control," ACM workshop on Cloud computing security (CCSW), 2009.

[65] M. Jakobsson and A. Juels, "Server-Side Detection of Malware Infection," in New Security Paradigms Workshop (NSPW), Oxford, UK, 2009.

[66] M. Jakobsson, "Captcha-free throttling," Proceedings of the 2nd ACM workshop on Security and artificial intelligence, 15–22, 2009.

[67] M. Jakobsson, E. Shi, P. Golle, and R. Chow, "Implicit authentication for mobile devices," Proceedings of the 4th USENIX conference on Hot topics in security, 9–9, 2009.

[68] C. Soghoian, O. Friedrichs and M. Jakobsson, "The Threat of Political Phishing," International Symposium on Human Aspects of Information Security & Assurance (HAISA 2008)

[69] R. Chow, P. Golle, M. Jakobsson, L. Wang and X. Wang, "Making CAPTCHAs Clickable," In proc. of HotMobile 2008.

[70] M. Jakobsson, A. Juels, and J. Ratkiewicz, "Privacy-Preserving History Mining for Web Browsers," Web 2.0 Security and Privacy, 2008.

[71] M. Jakobsson, E. Stolterman, S. Wetzel, L. Yang, "Love and Authentication," (Notes) ACM Computer/Human Interaction Conference (CHI), 2008. Also see `www.I-forgot-my-password.com`

[72] M. Jakobsson and S. Myers, "Delayed Password Disclosure," Proceedings of the 2007 ACM workshop on Digital Identity Management

[73] M. Jakobsson, S. Stamm, Z. Ramzan, "JavaScript Breaks Free," W2SP '07

[74] A. Juels, S. Stamm, M. Jakobsson, "Combatting Click Fraud via Premium Clicks," USENIX Security 2007

[75] R. Chow, P. Golle, M. Jakobsson, X. Wang, "Clickable CAPTCHAs," Ad-Fraud '07 Workshop; 2007 September 14; Stanford, CA, USA

[76] S. Stamm, Z. Ramzan, and M. Jakobsson, "Drive-by Pharming," In Proceedings of Information and Communications Security, 9th International Conference, ICICS 2007

[77] M. Jakobsson, A. Tsow, A. Shah, E. Blevis, Y.-K. Lim, "What Instills Trust? A Qualitative Study of Phishing," USEC '07.

[78] R. Akavipat, V. Anandpara, A. Dingman, C. Liu, D. Liu, K. Pongsanon, H. Roinestad and M. Jakobsson, "Phishing IQ Tests Measure Fear, not Ability," USEC '07.

[79] M. Jakobsson, "The Human Factor in Phishing," American Conference Institute's Forum on Privacy & Security of Consumer Information, 2007

[80] S. Srikwan, M. Jakobsson, A. Albrecht and M. Dalkilic, "Trust Establishment in Data Sharing: An Incentive Model for Biodiversity Information Systems," TrustCol 2006

[81] J.Y. Choi, P. Golle, M. Jakobsson, "Tamper-Evident Digital Signatures: Protecting Certification Authorities Against Malware," DACS '06

[82] L. Yang, M. Jakobsson, S. Wetzel, "Discount Anonymous On Demand Routing for Mobile Ad hoc Networks," SECURECOMM '06

[83] P. Golle, X. Wang, M. Jakobsson, A. Tsow, "Deterring Voluntary Trace Disclosure in Re-encryption Mix Networks." IEEE S&P '06

[84] M. Jakobsson, A. Juels, T. Jagatic, "Cache Cookies for Browser Authentication (Extended Abstract)," IEEE S&P '06

[85] M. Jakobsson and J. Ratkiewicz, "Designing Ethical Phishing Experiments: A study of (ROT13) rOnl auction query features.", WWW '06

[86] M. Jakobsson and S. Stamm. "Invasive Browser Sniffing and Countermeasures," WWW '06

[87] J.Y. Choi, P. Golle and M. Jakobsson. "Auditable Privacy: On Tamper-Evident Mix Networks," Financial Crypto '06

[88] A. Juels, D. Catalano and M. Jakobsson. "Coercion-Resistant Electronic Elections," WPES '05

[89] V. Griffith and M. Jakobsson. "Messin' with Texas, Deriving Mother's Maiden Names Using Public Records," ACNS '05, 2005.

[90] M. Jakobsson and L. Yang. "Quantifying Security in Hybrid Cellular Networks," ACNS '05, 2005

[91] Y.-C. Hu, M. Jakobsson, and A. Perrig. "Efficient Constructions for One-way Hash Chains," ACNS '05, 2005

[92] M. Jakobsson. "Modeling and Preventing Phishing Attacks," Phishing Panel in Financial Cryptography '05. 2005, abstract in proceedings.

[93] N. Ben Salem, J.-P. Hubaux, and M. Jakobsson. "Reputation-based Wi-Fi Deployment Protocols and Security Analysis," In WMASH '04. ACM Press, 2004. pp. 29–40.

[94] M. Jakobsson and S. Wetzel. "Efficient Attribute Authentication with Applications to Ad Hoc Networks," In VANET '04. ACM Press, 2004. pp. 38–46.

[95] M. Jakobsson, X. Wang, and S. Wetzel. "Stealth Attacks in Vehicular Technologies," Invited paper. In Proceedings of IEEE Vehicular Technology Conference 2004 Fall (VTC-Fall 2004). IEEE, 2004.

[96] A. Ambainis, H. Lipmaa, and M. Jakobsson. "Cryptographic Randomized Response Technique," In PKC '04. LNCS 2947. Springer-Verlag, 2004. pp. 425–438.

[97] P. Golle, M. Jakobsson, A. Juels, and P. Syverson. "Universal Re-encryption for Mixnets," In CT-RSA '04. LNCS 2964. Springer-Verlag, 2004. pp. 163–178.

[98] P. Golle and M. Jakobsson. "Reusable Anonymous Return Channels," In WPES '03. ACM Press, 2003. pp. 94–100.

[99] M. Jakobsson, S. Wetzel, B. Yener. "Stealth Attacks on Ad-Hoc Wireless Networks," In IEEE VTC '03, 2003.

[100] N. Ben Salem, L. Buttyan, J.-P. Hubaux, and M. Jakobsson. "A Charging and Rewarding Scheme for Packet Forwarding in Multi-hop Cellular Networks," In ACM MobiHoc '03. ACM Press, 2003. pp. 13–24.

[101] M. Jakobsson, J.-P.Hubaux and L. Buttyan. "A Micro-Payment Scheme Encouraging Collaboration in Multi-Hop Cellular Networks," In FC '03. LNCS 2742. Springer-Verlag, 2003. pp. 15–33.

[102] M. Jakobsson, T. Leighton, S. Micali and M. Szydlo. "Fractal Merkle Tree Representation and Traversal," In RSA-CT '03 2003.

[103] A. Boldyreva and M Jakobsson. "Theft protected proprietary certificates," In DRM '02. LNCS 2696, 2002. pp. 208–220.

[104] P. Golle, S. Zhong, M. Jakobsson, A. Juels, and D. Boneh. "Optimistic Mixing for Exit-Polls," In Asiacrypt '02. LNCS 2501. Springer-Verlag, 2002. pp. 451–465.

[105] P. MacKenzie, T. Shrimpton, and M. Jakobsson. "Threshold Password-Authenticated Key Exchange," In CRYPTO '02. LNCS 2442. Springer-Verlag, 2002. pp. 385–400.

[106] M. Jakobsson. "Fractal Hash Sequence Representation and Traversal," In Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT '02). 2002. pp. 437–444.

[107] M. Jakobsson, A. Juels, and R. Rivest. "Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking," In Proceedings of the 11th USENIX Security Symposium. USENIX Association, 2002. pp. 339–353.

[108] D. Coppersmith and M. Jakobsson. "Almost Optimal Hash Sequence Traversal," In Financial Crypto '02. 2002.

[109] M. Jakobsson. "Financial Instruments in Recommendation Mechanisms," In Financial Crypto '02. 2002.

[110] J. Garay, and M. Jakobsson. "Timed Release of Standard Digital Signatures," In Financial Crypto '02. 2002.

[111] F. Menczer, N. Street, N. Vishwakarma, A. Monge, and M. Jakobsson. "Intellishopper: A Proactive, Personal, Private Shopping Assistant," In AAMAS '02. ACM Press, 2002. pp. 1001–1008.

[112] M. Jakobsson, A. Juels, and P. Nguyen. "Proprietary Certificates," In CT-RSA '02. LNCS 2271. Springer-Verlag, 2002. pp. 164–181.

[113] M. Jakobsson and A. Juels. "An Optimally Robust Hybrid Mix Network," In PODC '01. ACM Press. 2001. pp. 284–292.

[114] M. Jakobsson and M. Reiter. "Discouraging Software Piracy Using Software Aging," In DRM '01. LNCS 2320. Springer-Verlag, 2002. pp. 1–12.

[115] M. Jakobsson and S. Wetzel. "Security Weaknesses in Bluetooth," In CT–RSA '01. LNCS 2020. Springer-Verlag, 2001. pp. 176–191.

[116] M. Jakobsson and D. Pointcheval. "Mutual Authentication for Low-Power Mobile Devices," In Financial Crypto '01. LNCS 2339. Springer-Verlag, 2001. pp. 178–195.

[117] M. Jakobsson, D. Pointcheval, and A. Young. "Secure Mobile Gambling," In CT–RSA '01. LNCS 2020. Springer-Verlag, 2001. pp. 110–125.

[118] M. Jakobsson and S.Wetzel. "Secure Server-Aided Signature Generation," In PKC '01. LNCS 1992. Springer-Verlag, 2001. pp. 383–401.

[119] M. Jakobsson and A. Juels. "Addition of ElGamal Plaintexts," In T. Okamoto, ed., ASIACRYPT '00. LNCS 1976. Springer-Verlag, 2000. pp. 346–358.

[120] M. Jakobsson, and A. Juels. "Mix and Match: Secure Function Evaluation via Ciphertexts," In ASIACRYPT '00. LNCS 1976. Springer-Verlag, 2000. pp. 162–177.

[121] R. Arlein, B. Jai, M. Jakobsson, F. Monrose, and M. Reiter. "Privacy-Preserving Global Customization," In ACM E-Commerce '00. ACM Press, 2000. pp. 176–184.

[122] C.-P. Schnorr and M. Jakobsson. "Security of Signed ElGamal Encryption," In ASIACRYPT '00. LNCS 1976. Springer-Verlag, 2000. pp. 73–89.

[123] P. Bohannon, M. Jakobsson, and S. Srikwan. "Cryptographic Approaches to Privacy in Forensic DNA Databases," In Public Key Cryptography '00. LNCS 1751. Springer-Verlag, 2000, pp. 373–390.

[124] J. Garay, M. Jakobsson, and P. MacKenzie. "Abuse-free Optimistic Contract Signing," In CRYPTO '99. LNCS 1666. Springer-Verlag, 1999. pp. 449–466.

[125] M. Jakobsson. "Flash Mixing," In PODC '99. ACM Press, 1999. pp. 83–89.

[126] G. Di Crescenzo, N. Ferguson, R. Impagliazzo, and M. Jakobsson. "How To Forget a Secret," In STACS '99. LNCS 1563. Springer-Verlag, 1999. pp. 500–509.

[127] M. Jakobsson, D. M'Raihi, Y. Tsiounis, and M. Yung. "Electronic Payments: Where Do We Go from Here?," In CQRE (Secure) '99. LNCS 1740. Springer-Verlag, 1999. pp. 43–63.

[128] C.P. Schnorr and M. Jakobsson. "Security Of Discrete Log Cryptosystems in the Random Oracle + Generic Model," In Conference on The Mathematics of Public-Key Cryptography. 1999.

[129] M. Jakobsson and A. Juels "Proofs of Work and Breadpudding Protocols," In CMS '99. IFIP Conference Proceedings, Vol. 152. Kluwer, B.V., 1999. pp. 252 – 272.

[130] M. Jakobsson and C-P Schnorr. "Efficient Oblivious Proofs of Correct Exponentiation," In CMS '99. IFIP Conference Proceedings, Vol. 152. Kluwer, B.V., 1999. pp. 71–86.

[131] M. Jakobsson, P. MacKenzie, and J.P. Stern. "Secure and Lightweight Advertising on the Web," In World Wide Web '99

[132] M. Jakobsson, J.P. Stern, and M. Yung. "Scramble All, Encrypt Small," In Fast Software Encryption '99. LNCS 1636. Springer-Verlag, 1999. pp. 95–111.

[133] M. Jakobsson and J. Mueller. "Improved Magic Ink Signatures Using Hints," In Financial Cryptography '99. LNCS 1648. Springer-Verlag, 1999. pp. 253–268.

[134] M. Jakobsson. "Mini-Cash: A Minimalistic Approach to E-Commerce," In Public Key Cryptography '99. LNCS 1560. Springer-Verlag, 1999. pp. 122–135.

[135] M. Jakobsson. "On Quorum Controlled Asymmetric Proxy Re-encryption," In Public Key Cryptography '99. LNCS 1560. Springer-Verlag, 1999. pp. 112–121.

[136] M. Jakobsson and A. Juels. "X-Cash: Executable Digital Cash," In Financial Cryptography '98. LNCS 1465. Springer-Verlag, 1998. pp. 16–27.

[137] M. Jakobsson and D. M'Raihi. "Mix-based Electronic Payments," In Proceedings of the Selected Areas in Cryptography. LNCS 1556. Springer-Verlag, 1998. pp. 157173.

[138] M. Jakobsson, E. Shriver, B. Hillyer, and A. Juels. "A Practical Secure Physical Random Bit Generator," In CCS '98: Proceedings of the 5th ACM conference on Computer and communications security. ACM Press, 1998. pp. 103–111.

[139] M. Jakobsson. "A Practical Mix," In Advances in Cryptology – EuroCrypt '98. LNCS 1403. Springer-Verlag, 1998. pp. 448–461.

[140] M. Jakobsson and M. Yung. "On Assurance Structures for WWW Commerce," In Financial Cryptography '98. LNCS 1465. Springer-Verlag, 1998. pp. 141–157.

[141] E. Gabber, M. Jakobsson, Y. Matias, and A. Mayer. "Curbing Junk E-Mail via Secure Classification," In Financial Cryptography '98. LNCS 1465. Springer-Verlag, 1998. pp. 198–213.

[142] M. Jakobsson and M. Yung. "Distributed 'Magic Ink' Signatures," In Advances in Cryptology – EuroCrypt '97. LNCS 1233. Springer-Verlag, 1997. pp. 450–464.

[143] M. Jakobsson and M. Yung. "Applying Anti-Trust Policies to Increase Trust in a Versatile E-Money System," In Financial Cryptography '97. LNCS 1318. Springer-Verlag, 1997. pp. 217–238.

[144] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. "Proactive public-key and signature schemes," In Proceedings of the 4th Annual Conference on Computer Communications Security. ACM Press, 1997. pp. 100–110.

[145] M. Bellare, M. Jakobsson, and M. Yung. "Round-Optimal Zero-Knowledge Arguments Based on any One-Way Function," In Advances in Cryptology – EuroCrypt '97. LNCS 1233. Springer-Verlag, 1997. pp. 280–305.

[146] M. Jakobsson and M. Yung. "Proving Without Knowing," In Crypto '96. LNCS 1109. Springer-Verlag, 1996. pp. 186–200.

[147] M. Jakobsson, K. Sako, and R. Impagliazzo. "Designated Verifier Proofs and Their Applications," In Advances in Cryptology – EuroCrypt '96. LNCS 1070. Springer-Verlag, 1996. pp. 143–154.

[148] M. Jakobsson and M. Yung. "Revokable and Versatile Electronic Money," In CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security. ACM Press, 1996. pp. 76–87.

[149] M. Jakobsson. "Ripping Coins for a Fair Exchange," In Advances in Cryptology – EuroCrypt '95. LNCS 921. Springer-Verlag, 1995. pp. 220–230.

[150] M. Jakobsson. "Blackmailing using Undeniable Signatures," In Advances in Cryptology EuroCrypt '94. LNCS 950. Springer-Verlag, 1994. pp. 425–427.

[151] M. Jakobsson. "Reducing costs in identification protocols," Crypto '92, 1992.

[152] M. Jakobsson. "Machine-Generated Music with Themes," In International Conference on Artificial Neural Networks '92. Vol 2. Amsterdam: Elsevier, 1992. pp. 1645–1646

[153] M. Jakobsson, "Social Engineering 2.0: What's Next," McAfee Security Journal, Fall 2008

[154] M. Jakobsson and S. Myers, "Delayed Password Disclosure," ACM SIGACT News archive, Volume 38, Issue 3 (September 2007), pp. 56 - 75

[155] V. Griffith and M. Jakobsson. "Messin' with Texas, Deriving Mother's Maiden Names Using Public Records," CryptoBytes, 2007.

[156] M. Jakobsson, A. Juels, J. Ratkiewicz, "Remote-Harm Detection," Beta-version available at `rhd.ravenwhitedevelopment.com/`

[157] S. Stamm, M. Jakobsson, "Social Malware," Experimental results available at `www.indiana.edu/ phishing/verybigad/`

[158] M. Jakobsson. "Privacy vs. Authenticity," Ph.D. Thesis, University of California at San Diego, 1997

[159] Markus Jakobsson, Automatic PIN creation using passwords, US20130125214 A1, 2012.

[160] Markus Jakobsson, Systems and methods for creating a user credential and authentication using the created user credential, US 20130111571 A1, 2012.

[161] Markus Jakobsson, Password check by decomposing password, US 20120284783 A1, 2012.

[162] Markus Jakobsson, William Leddy, System and methods for protecting users from malicious content, US 20120192277 A1, 2011.

[163] Markus Jakobsson, Karl-Anders R. Johansson, Auditing a device, US8370935 B1, 2011.

[164] Markus Jakobsson, Methods and Apparatus for Efficient Computation of One-Way Chains in Cryptographic Applications, US20120303969 A1, 2011.

[165] Markus Jakobsson, Automatic PIN creation using password, US 20120110634 A1, 2011.

[166] Markus Jakobsson, Jim Roy Palmer, Gustavo Maldonado, Interactive CAPTCHA, US20130007875 A1, 2011.

[167] Markus Jakobsson, System access determination based on classification of stimuli, US 20110314559 A1, 2011.

[168] Markus Jakobsson, System access determination based on classification of stimuli, WO 2011159356 A1, 2011.

[169] Markus Jakobsson, Method, medium, and system for reducing fraud by increasing guilt during a purchase transaction, US8458041 B1, 2011.

[170] Markus Jakobsson, Visualization of Access Information, US 20120233314 A1, 2011.

[171] Markus Jakobsson, Richard Chow,Runting Shi, Implicit authentication, US20120137340 A1, 2010.

[172] Markus Jakobsson, Karl-Anders R. Johansson, Auditing a device, EP2467793 A1, 2010.

[173] Markus Jakobsson, Event log authentication using secure components, US 20110314297 A1, 2010.

[174] Markus Jakobsson, Philippe J.P. Golle, Risk-based alerts, US 20110314426 A1, 2010.

[175] Markus Jakobsson, Karl-Anders R. Johansson, Auditing a device, US 20110041178 A1, 2010.

[176] M. Jakobsson, A. Juels, J. Kaliski Jr, S. Burton and others, Identity authentication system and method, US Patent 7,502,933, 2009.

[177] Markus Jakobsson, Method and system for facilitating throttling of interpolation-based authentication, US8219810 B2, 2009.

[178] Markus Jakobsson, Karl-Anders R. Johansson, Auditing a device, US8375442 B2, 2009.

[179] Markus Jakobsson, Karl-Anders R. Johansson, Auditing a device, US 20110041180 A1, 2009.

[180] Markus Jakobsson, Pattern-based application classification, US 20110055925 A1, 2009.

[181] Markus Jakobsson, Method and apparatus for detecting cyber threats, US8286225 B2, 2009.

[182] Markus Jakobsson, Method and apparatus for detecting cyber threats, US20110035784 A1, 2009.

[183] Markus Jakobsson, CAPTCHA-free throttling, US8312073 B2, 2009.

[184] Markus Jakobsson, Captcha-free throttling, US 20110035505 A1, 2009.

[185] Markus Jakobsson, Christopher Soghoian, Method and apparatus for throttling access using small payments, US 20100153275 A1, 2008.

[186] Markus Jakobsson, Christopher Soghoian, Method and apparatus for mutual authentication using small payments, US 20100153274 A1, 2008.

[187] Philippe J.P. Golle, Markus Jakobsson,Richard Chow, Resetting a forgotten password using the password itself as authentication, US 20100125906 A1, 2008.

[188] Philippe J. P. Golle, Markus Jakobsson,Richard Chow, Authenticating users with memorable personal questions, US8161534 B2, 2008.

[189] Richard Chow, Philippe J.P. Golle, Markus Jakobsson, Jessica N. Staddon, Authentication based on user behavior, US20100122329 A1, 2008.

[190] Richard Chow, Philippe J.P. Golle, Markus Jakobsson, Jessica N. Staddon, Enterprise password reset, US 20100122340 A1, 2008.

[191] Markus Jakobsson, Methods and apparatus for efficient computation of one-way chains in cryptographic applications, US8086866 B2, 2008.

[192] Richard Chow, Philippe J. P. Golle, Markus Jakobsson, Selectable captchas, US8307407 B2, 2008.

[193] Markus Jakobsson, Ari Juels, Sidney Louis Stamm, Method and apparatus for combatting click fraud, US 20080162227 A1, 2007.

[194] Jakobsson, Method and apparatus for evaluating actions performed on a client device, US 20080037791 A1, 2007.

[195] Jakobsson, Ari Juels, Method and apparatus for storing information in a browser storage area of a client device, US 20070106748 A1, 2006.

[196] Markus Jakobsson, Steven Andrew Myers, Anti-phising logon authentication object oriented system and method, WO 2006062838 A1, 2005.

[197] Jakobsson, Jean-Pierre Hubaux,Levente Buttyan, Micro-payment scheme encouraging collaboration in multi-hop cellular networks, US 20050165696 A1, 2004.

[198] Andrew Nanopoulos, Karl Ackerman, Piers Bowness, William Duane, Markus Jakobsson, Burt Kaliski, Dmitri Pal, Shane D. Rice,Less , System and method providing disconnected authentication, WO2005029746 A3, 2004.

[199] Andrew Nanopoulos, Karl Ackerman, Piers Bowness, William Duane, Markus Jakobsson, Burt Kaliski, Dmitri Pal, Shane Rice, Ronald Rivest, Less , System and method providing disconnected authentication, US 20050166263 A1, 2004.

[200] Andrew Nanopoulos, Karl Ackerman, Piers Bowness, William Duane, Markus Jakobsson, Burt Kaliski, Dmitri Pal, Shane D. Rice,Less , Systeme et procede d'authentification deconnectee, WO 2005029746 A2, 2004.

[201] Markus Jakobsson, Ari Juels, Burton S. Kaliski Jr., Identity authentication system and method, WO2004051585 A3, 2003.

[202] Markus Jakobsson, Ari Juels, Burton S. Kaliski, Jr., Identity authentication system and method, US 7502933 B2, 2003.

[203] Markus Jakobsson, Ari Juels, Burton S. Kaliski Jr., Systeme et procede de validation d'identite, WO 2004051585 A2, 2003.

[204] Markus Jakobsson, Burton S. Kaliski, Jr., Method and apparatus for graph-based partition of cryptographic functionality, US7730518 B2, 2003.

[205] Markus Jakobsson, Phong Q. Nguyen, Methods and apparatus for private certificates in public key cryptography, US7404078 B2, 2002.

[206] Jakobsson, Philip MacKenzie, Thomas Shrimpton, Method and apparatus for performing multi-server threshold password-authenticated key exchange, US20030221102 A1, 2002.

[207] Markus Jakobsson, Philip D MacKenzie, Method and apparatus for distributing shares of a password for use in multi-server password authentication, US 7073068 B2, 2002.

[208] Markus Jakobsson, Methods and apparatus for efficient computation of one-way chains in cryptographic applications, EP 1389376 A1 (text from WO2002084944A1), 2002.

[209] Markus Jakobsson, Adam Lucas Young, Method and apparatus for identification tagging documents in a computer system, US 7356845 B2, 2002.

[210] Juan A. Garay, Markus Jakobsson, Methods and apparatus for computationally-efficient generation of secure digital signatures, US 7366911 B2, 2001.

[211] Markus Jakobsson, Methods and apparatus for efficient computation of one-way chains in cryptographic applications, US 7404080 B2, 2001.

[212] Markus Jakobsson, Susanne Gudrun Wetzel, Securing the identity of a bluetooth master device (bd addr) against eavesdropping by preventing the association of a detected channel access code (cac) with the identity of a particular bluetooth device, WO2002019641 A3, 2001.

[213] Markus Jakobsson, Susanne Gudrun Wetzel, Procede et appareil permettant d'assurer la securite d'utilisateurs de dispositifs a capacites bluetooth, WO 2002019641 A2, 2001.

[214] Robert M. Arlein, Ben Jai, Markus Jakobsson, Fabian Monrose, Michael Kendrick Reiter, Less , Methods and apparatus for providing privacy-preserving global customization, US 7107269 B2, 2001.

[215] Markus Jakobsson, Susanne Gudrun Wetzel, Secure distributed computation in cryptographic applications, US6950937 B2, 2001.

[216] Markus Jakobsson, Susanne Gudrun Wetzel, Method and apparatus for ensuring security of users of short range wireless enable devices, US6981157 B2, 2001.

[217] Markus Jakobsson, Susanne Gudrun Wetzel, Method and apparatus for ensuring security of users of bluetooth TM-enabled devices, US 6574455 B2, 2001.

[218] Garay; Juan A. (West New York, NJ), Jakobsson; M. (Hoboken, NJ), Kristol; David M. (Summit, NJ), Mizikovsky; Semyon B.(Morganville, NJ), Cryptographic key processing and storage , 7023998, 2001.

[219] Markus Jakobsson, Method, apparatus, and article of manufacture for generating secure recommendations from market-based financial instrument prices, US 6970839 B2, 2001.

[220] Markus Jakobsson, Encryption method and apparatus with escrow guarantees, US7035403 B2, 2001.

[221] Jakobsson, Call originator access control through user-specified pricing mechanism in a communication network, US20020099670 A1, 2001.

[222] Markus Jakobsson, Claus Peter Schnorr, Tagged private information retrieval, US7013295 B2, 2000.

[223] Markus Jakobsson, Secure enclosure for key exchange, US 7065655 B1, 2000.

[224] Markus Jakobsson, Michael Kendrick Reiter, Software aging method and apparatus for discouraging software piracy, US 7003110 B1, 2000.

[225] Markus Jakobsson, Probabilistic theft deterrence, US6501380 B1, 2000.

[226] Markus Jakobsson, Michael Kendrick Reiter, Abraham Silberschatz, Anonymous and secure electronic commerce, EP 1150227 A1, 2000.

[227] Markus Jakobsson, Ari Juels, Proofs of work and bread pudding protocols, US 7356696 B1, 2000.

[228] Markus Jakobsson, Joy Colette Mueller, Methods of protecting against spam electronic mail, US7644274 B1, 2000.

[229] Philip L. Bohannon, Markus Jakobsson,Fabian Monrose, Michael Kendrick Reiter,Susanne Gudrun Wetzel, Less , Generation of repeatable cryptographic key based on varying parameters, EP1043862 B1, 2000.

[230] Markus Jakobsson, Ari Juels, Mix and match: a new approach to secure multiparty computation, US6772339 B1, 2000.

[231] Markus Jakobsson, Ari Juels, Mixing in small batches, US6813354 B1, 2000.

[232] Philip L. Bohannon, Markus Jakobsson,Fabian Monrose, Michael Kendrick Reiter,Susanne Gudrun Wetzel, Less , Generation of repeatable cryptographic key based on varying parameters, US 6901145 B1, 36566

[233] Markus Jakobsson, Claus Peter Schnorr, Non malleable encryption method and apparatus using key-encryption keys and digital signature, US6931126 B1, 2000.

[234] Markus Jakobsson, Claus Peter Schnorr, Non malleable encryption method and apparatus using key-encryption keys and digital signature, US 6931126 B1, 2000.

[235] Markus Jakobsson, Flash mixing apparatus and method, US 6598163 B1, 1999.

[236] Markus Jakobsson, Minimalistic electronic commerce system, US 6529884 B1, 1999.

[237] Markus Jakobsson, Method and system for providing translation certificates, US 6687822 B1, 1999.

[238] Markus Jakobsson, Verification of correct exponentiation or other operations in cryptographic applications, US6978372 B1, 1999.

[239] Markus Jakobsson, Non malleable encryption apparatus and method, US 6507656 B1, 1999.

[240] Markus Jakobsson, Method and system for quorum controlled asymmetric proxy encryption, US 6587946 B1, 1998.

[241] Markus Jakobsson, Practical mix-based election scheme, US 6317833 B1, 1998.

[242] Markus Jakobsson, Ari Juels, Method and apparatus for extracting unbiased random bits from a potentially biased source of randomness, US 6393447 B1, 1998.

[243] Markus Jakobsson, Ari Juels, Executable digital cash for electronic commerce, US6157920 A, 1998.

[244] Bruce Kenneth Hillyer, Markus Jakobsson, Elizabeth Shriver, Storage device random bit generator, US 6317499 B1, 1998.

[245] Markus Jakobsson, Method and apparatus for encrypting, decrypting, and providing privacy for data values, US 6049613 A, 1998.