UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

BLUE COAT SYSTEMS LLC,
Petitioner,

v.

FINJAN, INC.,
Patent Owner.

Patent No. 9,219,755

DECLARATION OF AZER BESTAVROS, Ph.D.

TABLE OF CONTENTS

I, Azer Bestavros, declare as follows:

I. QUALIFICATIONS

1. I am a Professor of Computer Science at Boston University, whose faculty I joined in 1991. I chaired the Computer Science Department from 2000 to 2007, overseeing a period of significant growth, culminating in the Chronicle of Higher Education's ranking of the Department as seventh in the U.S. in terms of scholarly productivity.

2. I am the Founding Director of the BU Hariri Institute for Computing at Boston University, which was set up in 2010 to "create and sustain a community of scholars who believe in the transformative potential of computational perspectives in research and education." I am also the Chair of the Data Science Initiative at Boston University.

3. In addition to my academic responsibilities at Boston University, over the years I have taken significant regional and national research leadership responsibilities. This includes: serving since 2010 as co-chair for Research, Education, and Outreach of the Massachusetts Green High-Performance Computing Center , a consortium of the five major research institutions in the Commonwealth of Massachusetts (Boston University, Harvard University, MIT, Northeastern University, and the University of Massachusetts); serving since 2013 as board member of the Cloud Computing Caucus, a non-profit, non-partisan

coalition of industry and key government stakeholders, focused on raising awareness and educating lawmakers and the public on issues associated with cloud computing; and serving from 2007 to 2012 as the elected chair of the IEEE Computer Society Technical Committee on the Internet.

4. I am a senior member of the Association for Computing Machinery (ACM) and a senior member of the Computer Society of the Institute of Electrical and Electronics Engineers (IEEE), among other professional societies and organizations. Within these organizations, I served as general chair, PC chair or PC member of most flagship technical conferences in networking, real-time systems, and databases. I co-organized formative workshops that led to ACM SIGPLAN LCTES and ACM SIGCOMM IMC. I have also served on the editorial board of major journals and periodicals, currently including IEEE Internet Computing and Communications of the ACM.

5. Prior to joining the faculty at Boston University, from June 1988 to September 1991, I was a Research Fellow, Teaching Fellow, and Research Assistant at Harvard University. From September 1985 to June 1987, I was a Research Assistant, Teaching Assistant, and Instructor at Alexandria University (Egypt).

6. I obtained my Ph.D. in Computer Science in 1992 from Harvard University under Thomas E. Cheatham, one of the “roots” of the academic

genealogy of applied computer scientists. I also hold a Master of Science degree in Computer Science from Harvard University, which I obtained in 1988; a Master of Science degree in Computer Science and Automatic Control from Alexandria University, which I obtained in 1987; and a Bachelor of Science degree in Computer Engineering from Alexandria University, which I obtained in 1984.

7. I have studied, taught, practiced, and conducted research in Computer Science and Computer Engineering for more than 30 years. My expertise is in the broad fields of computer networking, distributed systems, and real-time computing, with significant experience in Web content caching and distribution systems, scalable Internet services, cloud computing, Internet architecture, and networking protocols, among others.

8. I have extensive consulting and industrial research experience, including past and current engagements with a number of technology firms, including BBN Technologies, Sycamore Networks, NetApp, Microsoft, Verizon Labs, Macromedia, Allaire, Bowne, SUTI Technologies, and AT&T Bell Labs. I have consulted and served on the technical advisory board of many companies, and I have been retained by a number of law firms as a consultant on intellectual property issues related to Internet technologies and applications.

9. My curricular development efforts include my CS-350 course, which I developed and have taught since 1998. Through a rigorous treatment of the

invariant concepts underlying computing systems design, CS-350 familiarizes students with canonical problems that reoccur in software systems, including operating systems, networks, databases, and distributed systems, and provides students with a set of classical algorithms and basic performance evaluation techniques for tackling such problems. More recently, I have spearheaded a team effort to develop a set of courses for non-majors that can be used to introduce elements of mathematical abstraction, quantitative and methodical thinking, as utilized in mathematics, statistics, and computer science, with an emphasis on their relevance in our daily lives as reflected in widely used Internet and Web technologies and applications. In addition to these courses, I have taught undergraduate courses and graduate seminars on large-scale Internet systems, sensor networks, computer architecture, and real-time systems, and have guest-lectured in Sociology on issues related to Technology, Society and Public Policy.

10. Over the years, my contributions in research, teaching, and service have been recognized by a number of awards, including multiple best-paper awards from IEEE and ACM conferences, multiple distinguished ACM and IEEE service awards, and being selected multiple times as a distinguished speaker of the IEEE Computer Society (most recently in 2010). In 2010, I received the United Methodist Scholar Teacher Award in recognition of "outstanding dedication and contributions to the learning arts and to the institution" at Boston University, and

the ACM Sigmetrics Inaugural Test of Time Award for research results "whose impact is still felt 10-15 years after its initial publication."

11. A copy of my Curriculum Vitae, attached as EX1003, contains further details on my education, experience, publications, patents, and other qualifications to render an expert opinion in this matter.

II. SCOPE OF WORK

12. I understand that a petition is being filed with the United States Patent and Trademark Office for *inter partes* review of U.S. Patent No. 9,219,755 to Shlomo Touboul ("the '755 Patent," attached as EX1001), entitled "Malicious Mobile Code Runtime Monitoring System and Methods."

13. I have been retained by Blue Coat Systems LLC ("Blue Coat") to offer an expert opinion on the patentability of the claims of the '755 patent, as well as several other patents assigned to Finjan. I receive $550 per hour for my services. No part of my compensation is dependent on my opinions or on the outcome of this proceeding. I have previously testified for Blue Coat as an expert on the issue of noninfringement in *Finjan, Inc. v. Blue Coat Systems, Inc.*, No. 5:13-cv-03999-BLF (N.D. Cal.), which involved different patents. I also provided testimonial evidence in *Blue Coat Systems LLC v. Finjan, Inc.*, Nos. IPR2016-01441, -01443, and -01444 (PTAB), which also involved different patents. I do not have any other current or past affiliation as an expert witness or consultant with Blue Coat.

14. I have been specifically asked to provide my opinions on claims 1-8 of the '755 patent. In connection with this analysis, I have reviewed the '755 patent and relevant parts of its file history. I have also reviewed and considered various other documents in arriving at my opinions, and may cite to them in this declaration. For convenience, a list of exhibits considered in arriving at my opinions is included as an appendix to this declaration.

III. OVERVIEW OF THE '755 PATENT

15. The '755 patent relates to systems for protecting one or more personal computers or other network accessible devices or processes from undesirable or otherwise malicious operations.

16. The '755 patent has two independent claims. Claims 1 and 3 are both directed to a system for reviewing an operating system call issued by a downloadable. Each independent claim recites closely related subject matter, including a processor and memory, an operating system probe, a runtime environment monitor, a response engine, a downloadable engine, a request broker, a file system probe, and a network system probe. The systems use the probes to detect behavior of a downloadable, compare that behavior to security policies using the runtime environment monitor, and uses the response engine to block calls that violate the security policy while allowing them otherwise.

17. The dependent claims of the '755 patent either specify the nature of the downloadable or specify additional actions taken by the response engine. For example, claims 2 and 4 recite that the Downloadable is a Java component, an ActiveX control, executable code, or interpretable code. Claims 5-8 specify that the response engine takes an additional action: in claim 5 the action includes storing comparison results in an event log, in claim 6 the action includes informing the user, in claim 7 the action includes storing information in a database, and in claim 8 the action includes discarding the downloadable.

18. The text of the '755 patent is virtually devoid of support for the subject matter to which its claims are directed. For example, nowhere does the specification of the '755 patent disclose an operating system probe, a runtime environment monitor, a response engine, or various other elements of the '755 patent claims. Instead, certain documents incorporated by reference into the '755 patent relate to the claimed subject matter. For example, U.S. Patent No. 6,167,520 ("the '520 patent," EX1008) discusses a system with operating system probes, a runtime environment monitor, and a response engine. *See, e.g.,* EX1008 at 2:1-23.

19. The '520 patent was filed on January 29, 1997, making it the earliest-filed non-provisional application to which the '755 patent claims priority. The '755 patent also claims priority to U.S. Provisional Application No. 60/030,639 ("the '639 provisional," EX1010), filed November 8, 1996. I have reviewed the '639

provisional, and it does not provide support for the claims of the '755 patent. For example, the '639 provisional does not describe an operating system probe, a response engine, or various other claim elements of the '755 patent.

20. Accordingly, I understand that the'755 patent is entitled to a priority date no earlier than January 29, 1997. Nonetheless, my opinions would not change if the priority date of the '755 patent were November 8, 1996.

IV. LEGAL STANDARDS

21. I have been advised that a claim is not patentable under 35 U.S.C. § 102[1] if the claimed invention is not new. For the claim to be unpatentable because it is anticipated, all of its requirements must have been described in a single prior art reference, such as a publication or patent that predates the claimed invention.

22. I have also been advised that the description in the prior art reference does not have to be in the same words as the claim, but all of the requirements of the claim must be there, either stated or necessarily implied, so that someone of ordinary skill in the relevant field looking at that one reference would be able to make and use the claimed invention.

[1] Citations are to the relevant pre-AIA sections of 35 U.S.C.

23. I have been advised that a claimed invention is not patentable under 35 U.S.C. § 103 if it is obvious. A patent claim is unpatentable if the claimed invention would have been obvious to a person of ordinary skill in the field at the time the claimed invention was made. This means that even if all of the requirements of the claim cannot be found in a single prior art reference that would anticipate the claim, a person of ordinary skill in the relevant field who knew about all this prior art would have come up with the claimed invention.

24. I have further been advised that the ultimate conclusion of whether a claim is obvious should be based upon several factual determinations. That is, a determination of obviousness requires inquiries into: (1) the level of ordinary skill in the field; (2) the scope and content of the prior art; (3) what difference, if any, existed between the claimed invention and the prior art; and (4) any secondary evidence bearing on obviousness.

25. I have been advised that, in determining the level of ordinary skill in the field that someone would have had at the time the claimed invention was made, I should consider: (1) the levels of education and experience of persons working in the field; (2) the types of problems encountered in the field; and (3) the sophistication of the technology.

26. I have also been advised that, in determining the scope and content of the prior art, in order to be considered as prior art, a reference must be reasonably

related to the claimed invention of the patent. A reference is reasonably related if it is in the same field as the claimed invention or is from another field to which a person of ordinary skill in the field would look to solve a known problem.

27. I have been advised that any secondary evidence of nonobviousness may be considered as an indication that the claimed invention would not have been obvious at the time the claimed invention was made, and any secondary evidence of obviousness may be considered as an indication that the claimed invention would have been obvious at such time. Although I should consider any such evidence, I should also assign it appropriate relevance and importance when deciding whether the claimed invention would have been obvious.

28. I have been advised that a patent claim composed of several elements is not proved obvious merely by demonstrating that each of its elements was independently known in the prior art. In evaluating whether such a claim would have been obvious, I may consider whether there is a reason that would have prompted a person of ordinary skill in the field to combine the elements or concepts from the prior art in the same way as in the claimed invention.

29. I have been further advised that there is no single way to define the line between true inventiveness on the one hand (which is patentable) and the application of common sense and ordinary skill to solve a problem on the other hand (which is not patentable). For example, market forces or other design

incentives may be what produced a change, rather than true inventiveness. I may consider whether the change was merely the predictable result of using prior art elements according to their known functions, or whether it was the result of true inventiveness. I may also consider whether there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent. I may consider whether the innovation applies a known technique that had been used to improve a similar device or method in a similar way. I may also consider whether the claimed invention would have been obvious to try, meaning that the claimed innovation was one of a relatively small number of possible approaches to the problem with a reasonable expectation of success by those skilled in the art.

30. I have also been advised, however, that I must be careful not to determine obviousness using the benefit of hindsight; many true inventions might seem obvious after the fact. I should put myself in the position of a person of ordinary skill in the field at the time the claimed invention was made and I should not consider what is known today or what is learned from the teaching of the patent.

V. LEVEL OF ORDINARY SKILL AND RELEVANT TIME

31. I have been advised that "a person of ordinary skill in the art" is a hypothetical person to whom one could assign a routine task with reasonable

confidence that the task would be successfully carried out. I have been advised that the relevant timeframe is prior to the relevant priority date, which I understand is January 29, 1997.

32. The relevant technology field for the '755 patent is software for protection from network-propagated malware, including access rights management. By virtue of my education, experience, and training, I am familiar with the level of skill in the art of the '755 patent prior to January 29, 1997.

33. In my opinion, a person of ordinary skill in the relevant field prior to January 29, 1997 would include someone who had, through education or practical experience, the equivalent of a bachelor's degree in computer science or a related field and an additional four years of work in the field of computer security.

34. A person of ordinary skill in the relevant field would have been aware of and would have been working with trends from the early-to-mid 1990s, including trends towards access rights control of downloaded executable code using security policies. I understand that the person of ordinary skill in the art is presumed to be aware of the pertinent art. I discuss some of the most relevant art below.

VI. CLAIM CONSTRUCTION

35. I have been advised that, in the present proceeding, the claims of the '755 patent should generally be given their ordinary and customary meaning,

which is the meaning that the claim would have to a person of ordinary skill in the art at the time of the invention. Importantly, the person of ordinary skill in the art is deemed to read the claim term not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the specification. I have followed these principles in my analysis throughout this declaration. I discuss some particular terms below.

"downloadable"

36. The '755 patent uses the term "downloadable" throughout the claims, as well as in the specification. U.S. Patent No. 6,092,194 (EX1009), one of the patents from which the '755 claims priority, states that a "Downloadable is an executable application program, which is downloaded from a source computer and run on the destination computer." EX1009 at 1:44-55. This agrees with the use of the term "downloadable" in the '755 patent. Accordingly, I interpret the term "downloadable," as used in the '755 patent to include "an executable application program, which is downloaded from a source computer and run on a destination computer."

"extension call"

37. I understand that during prosecution, the Examiner objected to the use of the term "extension call" in the claims of the '755 patent, as the meaning of this term was unclear. EX1004 at 1485. In response, the applicant pointed to portions

of U.S. Patent No. 6,480,962 (the '962 Patent, EX1013) as allegedly providing support. EX1004 at 1601-04. These sections included Figs. 3 and 4, as well as part of the specification. *Id*. For example, the specification of the '962 Patent states,

> The security system 135a includes Java™ class extensions 304, wherein each extension 304 manages a respective one of the Java™ classes 302. When a new applet requests the service of a Java class 302, the corresponding Java™ class extension 304 interrupts the request and generates a message to notify the request broker 306 of the Downloadable's request.

EX1013 at 4:10-17. With regard to the classes the extension manages, the '962 patent states that they are "conventional Java™ classes 302. Each of the Java™ classes 302 performs a particular service such as loading applets, managing the network, managing file access, etc." EX1013 at 3:52-63.

38. As described in the '962 patent, a call is made by an applet to request the service of a Java class. There is a corresponding Java class extension that interrupts the request, and sends a message to a request broker. Claims 1 and 3 state that the extension call is part of a request made by the downloadable, and that the request is intercepted by the downloadable engine, so the extension call would be part of a request (such as a request for access to a file or network) from a downloadable, corresponding to an instance of a class, that is received by the downloadable engine (*e.g.*, a request from a Java applet handled by the Java

Virtual Machine). For the purposes of this declaration, I interpret the term "extension call" to at least include "a request corresponding to a class of program objects."

"downloadable engine"

39. The '755 patent uses the term "downloadable engine" in both independent claims. I have reviewed the specification of the '755 patent, and no mention is made of such a structure. I have also reviewed the specification of the '520 patent (EX1008) to which the '755 patent claims priority. The '962 patent referred to by the applicant during prosecution is a continuation of the '520 patent, sharing the same figures and specification. The '520 patent describes a "browser 245 [that] includes a Downloadable engine 250 for managing and executing received Downloadables." EX1008 at 3:29-30. The '520 patent gives the Java virtual machine and the ActiveX platform as examples of downloadable engines. *Id*. at 3:42-46, 5:17-19. Accordingly, I interpret the term "downloadable engine," as used in the '755 patent to include "software for managing and executing downloadables."

VII. OVERVIEW OF THE PRIOR ART

40. In the early- to mid-1990s, the popularity of the internet as a communication medium increased rapidly. At the same time, it became increasingly appreciated that large computer networks, such as the internet, posed a

danger, because files distributed over such networks could contain malicious or otherwise dangerous code. In fact, the "Morris Worm," arguably the first internet-propagating malware, was "invented" in 1988 by Robert Morris in an office across the hall from mine at Harvard. As the threat of malware was increasingly appreciated, this led to a proliferation of new software intended to combat this danger. *See, e.g.*, EX1012 at 12:20-50, Table 2 col. 5 (comparing an antivirus product to "four other shareware and commercial virus detection products"). Accordingly, by 1996, the market for antivirus protection was a crowded one, and the subject of a great deal of continuing research and development.

41. One common strategy for virus protection at the time was to enforce a system of access rights according to a security policy. By monitoring the behavior of a potentially dangerous program (such as one obtained from an untrusted source on the internet), and determining whether that program attempted an operation that exceeded its access rights, potentially malicious software could be detected and prevented from doing harm. The claims of the '755 patent are directed to systems that perform such methods. But as discussed below, such methods were already known to individuals of skill in the field as of 1996.

42. In my opinion, and as explained in further detail below, claims 1-8 of the '755 patent fail to identify anything new or significantly different from what

was already known to individuals of skill in the field prior to the filing of the application that led to the '755 patent, including prior to January 29, 1997.

43. Below is an overview of certain of the main prior art references that I rely on for my opinion that claims 1-8 of the '755 patent are unpatentable: Jaeger, CORBA, TBAV, and Arnold.

A. Jaeger

44. "Building Systems that Flexibly Control Downloadable Executable Content," by Jaeger *et al.* ("Jaeger," EX1005) was published in the Proceedings of the Sixth USENIX UNIX Security Symposium in July 1996.

45. I have personally attended USENIX conferences many times throughout my career, including several conferences in which I have presented a paper that was published in the proceedings. It has consistently been the practice of USENIX to publish the proceedings, including all papers submitted for the conference, in paper form as well as online, no later than the first day of the conference. EX1019 shows the Jaeger reference as it appeared in the Proceedings of the Sixth USENIX UNIX Security Symposium. *See* EX1019 at 131-48. Submissions are generally solicited for publication many months before the conference takes place, with final versions for publication due a month or more in advance, to ensure that the proceedings are available to participants at the conference.

46. Computer scientists routinely attend conferences like the USENIX conference in order to be able to review and discuss the presented papers, in the course of their own work and to keep abreast of developments in the field. Registration for such conferences typically requires a fee, but is generally open to any interested member of the public. The USENIX Security Symposium is a highly-reputable and competitive place to publish research publications, and would commonly be attended by a large number of computer scientists interested in the field.

47. The proceedings of USENIX symposia also remain available online and through mail afterwards. For example, the Proceedings of the Sixth USENIX UNIX Security Symposium were made available online for download in PostScript format in 1996 at usenix.org. *See* EX1020 (Archived Nov. 14, 1996, https://web.archive.org/web/19961114114919/http://usenix.org/publications/library/proceedings/sec96/jaeger.html). They were also available by mail for a small fee. *See* EX1021 (Archived Nov. 22, 1996 https://web.archive.org/web/19961122181208/http://usenix.org/publications/ordering/order.txt). The electronic version available online (EX1005) contains the same disclosure as the paper version (EX1019) provided to participants at the USENIX conference. *Compare* EX1005 *with* EX1019.

48. Accordingly, I have been advised that Jaeger is prior art under 35 U.S.C. § 102(a) because it was published in July 1996, which is prior to the January 29, 1997 effective filing date of the '755 patent.

49. Jaeger discloses an architecture for "flexible control" of downloaded executable content. EX1005 at 0002. The architecture provides that "downloaded executable content is properly controlled," thereby preventing a scenario in which "a malicious remote principal [*e.g.*, a user, group, service, etc.] may obtain unauthorized access to the downloading principal's resources." *Id*. Jaeger discloses that "operations are implemented by a call to authorize which checks the content access rights prior to calling the actual operation. Since these commands are only available in the browser and are protected by the authorization operation, only authorized accesses to system objects are possible." *Id*. at 0016. To this end, Jaeger describes "application specific interpreters [that] control access to application objects and determine the access rights of content within their domain." *Id*. at Fig. 3.

B. CORBA

50. "The Common Object Request Broker: Architecture and Specification" ("CORBA," EX1011) was published as a joint publication of the Object Management Group and X/Open in 1993. CORBA states on its face that it was published on December 29, 1993 as revision 1.2, OMG Document Number

93.xx.yy. Although CORBA identifies itself as a draft, this is because the CORBA specification was evolving at the time. CORBA was, however, widely distributed. *See* http://www.omg.org/gettingstarted/history_of_corba.htm. The Object Management Group website maintains a copy of CORBA, identified as document number 1993/93-12-43, for download in PDF or PostScript formats at http://www.omg.org/spec/CORBA/1.2/. From 1993 on, CORBA was widely distributed, including over the internet. For example, U.S. Patent No. 5,857,102 to McChesney *et al*. (EX1015), filed on March 14, 1995, states that "The Common Object Request Broker: Architecture and Specification, Rev. 1.2, OMG TC Document Number 93-12-43 [is] available by anonymous ftp to omg.org." EX1015 at 4:27-31; *see also* EX1023 at 2:20-25 (U.S. Patent No. 5,819,093, identifying the same document as "OMG Document Number 93.xx.yy Revision 1.2 dated 29 Dec. 1993 titled 'The Common Object Request Broker: Architecture and Specification' (otherwise referred to as CORBA) which is incorporated herein by reference"). Accordingly, I understand that CORBA is prior art under 35 U.S.C. § 102(a) or § 102(b) because it was published more than a year prior to the January 29, 1997 effective filing date of the '755 patent.

51. CORBA describes an architecture for distributed computing that uses an Object Request Broker ("ORB") to provide interoperability between applications on different machines. EX1011 at 15. The ORB receives a request

from a client and forwards the request to an object implementation. *Id*. at 30, Fig. 2. The ORB thus provides an interface for requesting execution of a code object that is "completely independent of where the object is located, what programming language it is implemented in, or any other aspect which is not reflected in the object's interface." *Id*. at 30. In other words, the ORB acts as a middleman between a client and an object requested for execution by the client.

C. TBAV

52. The user manual for the ThunderBYTE antivirus scanner ("TBAV," EX1006) was published in 1995. The ThunderBYTE antivirus scanner was a popular antivirus program in the mid-1990s, and TBAV was distributed as a printed publication along with the ThunderBYTE antivirus scanner software. *See, e.g.*, EX1017 ("Qualified organizations, editors, reviewers or PC users groups who prefer to receive a disk-based evaluation version of our software complete with a printed manual, please e-mail us or contact one of our local distributors." https://web.archive.org/web/19961204173839/http://thunderbyte.com/eval_req.html, archived Dec. 4, 1996); *see also* EX1018 at 1 (showing page offering an evaluation version of the software along with the manual ("/eval_req.html") was visited nearly 1,000 times per day in April of 1996).

53. Distribution of TBAV to the public is also demonstrated by U.S. Patent No. 5,696,822 (EX1012, the '822 patent), filed September 28, 1995. The

’822 patent lists the TBAV manual on its face and refers to the ThunderBYTE scanner in a list of “shareware and commercial virus detection products.” EX1012 at 12:20-50, Table 2 col. 5.

54. Accordingly, I have been advised that TBAV is prior art under 35 U.S.C. § 102(a) or § 102(b) because it was published in 1995, more than a year prior to the January 29, 1997 effective filing date of the ’755 patent.

55. TBAV describes a software program that protects computer systems against both known and unknown viruses. The TBAV software includes a wide range of utilities, including TbScan, a virus scanner; TbScanX, a memory resident version of TbScan; TbMem, a memory safeguard; TbFile, a file protector; TbDisk, a disk protector; and TbLog, a log file generator. EX1006 at 1-5, 139. The TbLog utility generates log files in response to various alert messages from TBAV virus detection utilities, including recording when a virus is detected. *Id*. at 139. The logs generated include information such as a time stamp, the machine where the event occurred, and a message describing what occurred and what files were involved. *Id*. The utility is “primarily for network users,” and allows a supervisor to “keep track of what’s going on.” *Id*.

a. Arnold

56. U.S. Patent No. 5,440,723 (“Arnold,” EX1007) was filed January 19, 1993 and published August 8, 1995. I have been advised that Arnold is prior art

under 35 U.S.C. §102(b) as it was patented more than 1 year before the January 29, 1997 effective filing date of the '755 patent.

57. Arnold discloses a system for detecting "anomalous behavior that may indicate the presence of an undesirable software entity such as a computer virus, worm, or Trojan Horse." EX1007 at Abstract. When such an entity is detected, Arnold's system extracts an identifying signature and adds the signature to a database so it may be used by a scanner to prevent subsequent infections of the system and other computers connected to the system in a network. *Id*.

VIII. GROUND 1: CLAIMS 1-4 ARE OBVIOUS OVER JAEGER IN VIEW OF CORBA

58. As explained in detail below, it is my opinion that each and every element of claims 1-4 of the '755 patent would have been obvious to a person of ordinary skill in the art in view of Jaeger and CORBA.

59. Each section of claims 1-4 of the '755 patent is presented below in bold text followed by my analysis of that part of the claim. The analysis below identifies exemplary disclosure of the cited references relative to the corresponding claim elements, and it is not meant to be exhaustive.

Claim 1, preamble: A system for reviewing an operating system call issued by a downloadable, comprising:

60. Jaeger discloses a system for reviewing an operating system call issued by a downloadable. Jaeger describes "Systems That Flexibly Control

Downloaded Executable Content." EX1005, Title. Jaeger describes the overall functions of its system as follows: "(1) authenticate the source of the content; (2) determine the least privilege access rights for the content given its source; and (3) enforce those access rights throughout the execution of the content." *Id.* at 0004. Jaeger describes the use of a "conventional protection model," in which "principals (e.g., users, groups, services, etc.) execute processes that perform operations (e.g., read, write, etc.) on objects (e.g., files, devices, etc.). The permissions of a principal to perform operations on objects are called the access rights of the principal. We call a set of access rights an access control domain or simply domain." EX1005 at 0003. Jaeger details the types of objects for which "access . . . needs to be controlled," including applications, file system objects, processes, network services, and the environment of each process. *Id.* at 0005. Accessing these object types involves calls to an operating system. Indeed, Jaeger specifies that the system identifies "I/O commands . . . to control access to system objects" and that the system includes an "operating system [that] has an unmodified trusted computing base, protects process domains, and provides authentication of principals." *Id.* at 0006-07. Accordingly, Jaeger discloses a system for reviewing an operating system call issued by a downloadable.

Claim 1.1: at least one processor for accessing elements stored in at least one memory associated with the at least one processor and for executing instructions associated with the elements, the elements including:

61. Jaeger's system includes a processor for accessing elements stored in at least one memory associated with the at least one processor and for executing instructions associated with the elements. Jaeger discloses "an architecture that flexibly controls the access rights of downloaded content." EX1005 at 0002. Jaeger describes this architecture as employing an "operating system [that] has an unmodified trusted computing base." *Id.* at 0006; *see also id.* at Fig. 2 (showing a system running on Network Computing Environments). A person of ordinary skill in the art reading Jaeger would understand that the computing base would include a processor and associated memory, for executing instructions.

62. Jaeger further discloses that:

> In general, access to the following types of objects needs to be controlled:
>
> - Application Objects: A process can read and write objects in its memory
> - File System: A process can perform read, write, and execute operations
> - Applications: A process can execute another application by creating a process for that application

Id. at 0005.

63. Accordingly, Jaeger teaches that the processor (running Jaeger's "process") is capable of accessing elements stored in the at least one memory (such as the "objects in its memory"), and that the processor executes the associated instructions. Furthermore, as Jaeger's architecture is intended to run on a computer, it would have been obvious to a person of ordinary skill to include a processor for executing instructions stored in associated memory, as such elements represent two of the essential building blocks for all of modern computer science. For similar reasons, a person of ordinary skill would have a reasonable expectation of success in making use of such elements. The system disclosed by Jaeger would be no exception; indeed, Jaeger states that the system's "model is influenced most strongly by the access control models of Hydra," which is a "kernel of a multiprocessor system." EX1005 at 0010, 0019.

Claim 1.2: an operating system probe associated with an operating system function for intercepting an operating system call being issued by a downloadable to an operating system and associated with the operating system function;

64. Jaeger also discloses an operating system probe associated with an operating system function for intercepting an operating system call being issued by a downloadable to an operating system and associated with the operating system function.

65. For example, Jaeger discloses that "[t]rusted, application-independent interpreters (which we will refer to as browsers from here on) control access to

system objects by the application-specific interpreters and downloaded content. An application-specific interpreter controls access to application objects and specifies the access rights of downloaded content within its limited domain." EX1005 at 0007. Jaeger describes the system as containing multiple interpreters (*e.g.*, the browser and each of the various application-specific interpreters), each including one or more probes, so that each different type of operation can be authorized and executed: "the browser determines which interpreter to execute the content. If the type refers to a known application-specific interpreter and the remote principal has the rights to execute content in that interpreter, then the content is run in that interpreter." *Id.* at 0007.

66. Jaeger describes the respective interpreters as being used to monitor different subsystems, including operating system functions. *Id.* at 0005. For example, Jaeger states that "access to the following types of objects needs to be controlled: . . . **Processes**: A process can communicate with another process via pipes or the file system." *Id.* (emphasis original). Managing processes is an example of an operating system function; indeed the '520 patent identifies "a process system probe" as an "operating system probe" that manages "instructions sent to the process system 275 of operating system 260." EX1008 at 4:10-23. Similarly, the remaining items in Jaeger's list include other operating system functions (e.g., Network and File Systems), and it would have been obvious to

include a probe for each of these functions, as Jaeger teaches that each one "needs to be controlled." *See* EX1005 at 0005.

67. Furthermore, Jaeger's probes intercept calls being issued by a downloadable as the downloadable is being executed.

> Content is executed as shown in Figure 5. . . . Any controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed. Operations are executed in the interpreter trusted with the operation.

EX1005 at 0008. If the interpreter authorizes controlled operations run by content, and then executes those requests, the interpreter's probes would be intercepting an operating system call being issued by a downloadable to an operating system and associated with the operating system function.

68. The protocol for content execution in Jaeger's system is shown in Fig. 5:

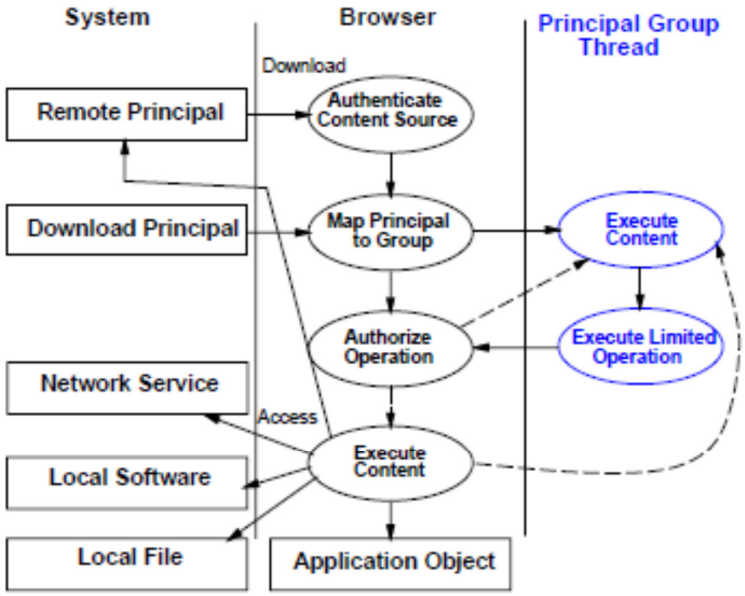


Figure 5: **Content Execution Protocol**: (1) Remote principal downloads content to browser; (2) Browser assigns content to principal group in application-specific interpreter (not shown); (3) content executes a limited operation which cause authorization in the browser; (4) if authorized the browser access system objects on behalf of content.

69. When a downloadable requests execution of an instruction by the operating system (*e.g.*, attempting to access one of the subsystems mentioned by Jaeger), this would constitute an "call being issued by a downloadable to an operating system and associated with the operating system function." By monitoring the operating system's subsystems for such requests, and intercepting those requests, Jaeger's interpreters act as operating system probes intercepting the calls.

70. Indeed, Jaeger's interpreters function in a manner that is substantially similar to the probes of the '520 patent. The '520 patent describes "operating

system probes" as respectively receiving from the downloadable "instructions sent to the file system … the network management system … the process system … [and] the memory system." EX1008 at 4:10-23; *see also id.* at Fig. 3 (showing the operating system probes situated between the operating system and the remainder of the security system, similar to Jaeger's interpreters). Jaeger's system includes interpreters that perform substantially the same function in substantially the same way.

71. As the interpreters of Jaeger receive requests and determine authorization status at the time of content execution, the probes intercept calls being made. Thus, Jaeger discloses an operating system probe associated with an operating system function for intercepting an operating system call being issued by a downloadable to an operating system and associated with the operating system function.

Claim 1.3: a runtime environment monitor for comparing the operating system call against a predetermined security policy including multiple security rules to determine if execution of the operating system call violates one or more of the multiple security rules before allowing the operating system to process the operating system call and for forwarding a message to a response engine when the comparison by the runtime environment monitor indicates a violation of one or more of the multiple security rules;

72. Jaeger teaches a runtime environment monitor for comparing the operating system call against a predetermined security policy including multiple security rules to determine if execution of the operating system call violates one or

more of the multiple security rules. Jaeger defines security policies according to "access rights." These access rights of a given correspond to the intersection of access rights the different system components being used. *See* EX1005 at 0012 ("The access rights of a remote principals content in interpreter t active at state s are the intersection of the rights of interpreter t and the rights of the principal group to which the remote principal is assigned in interpreter t at state s.") These rights are specified in advance because, as Jaeger describes, "we want delegation to be implicit relative to some application rather than require the downloading principal to specify access rights at runtime." *Id*. at 0014.

```
operation(args)
   args is a list of operation arguments
   arg_types is a list of the types of args
   arg_ops is a list of the ops to be run on args

   arg_types = types(args)
   If operation is not mandatory
      then collect predicate(operation, first(args))
         into arg_ops
   Foreach arg in args
      collect access predicates(operation, arg)
         into arg_ops
   If (authorize args arg_types arg_ops)
      then results = execute(procedure, args)
   Foreach transform in transforms
      apply-trans(transform,args,results)
```

Figure 8: Generated operation with authorization and transformation calls

73. Jaeger provides pseudocode in Figs. 8 and 9 illustrating how a requested operation can be authorized and subsequently executed subject to that authorization. Fig. 8 (above) illustrates a process for handling an operation. A request for an operation is received, including one or more arguments, "args." The types of each argument and the predicate operations ("ops") required for the requested operation each argument thereof are determined and stored as "arg_types" and "arg_ops," respectively. The operation process is subsequently suspended in the line reading "If (authorize args arg_types arg_ops)." This line calls the "authorize" procedure, which is illustrated in Fig. 9 (below). The "authorize" procedure returns a value of TRUE (FALSE) if the operation is permitted (forbidden) by the security policy. Accordingly, Jaeger's system compares an operating system call to a predetermined security policy and determines whether or not the policy is violated.

```
authorize(args arg_types arg_ops)
   For arg in args, arg_type in arg_types,
      ops in arg_ops
      Find object group for arg and arg_type
      For op in ops
         If domain rights grant op
            AND no exception precludes op on arg
            then continue
            else return FALSE
   return TRUE
```

Figure 9: Authorization procedure

74. Furthermore, Jaeger's system performs the comparison before allowing the operating system to process the operating system call. As shown in Fig. 8 of Jaeger, the "operation" function does not execute the procedure requested by the call until after evaluating whether it is authorized. Indeed, the procedure is executed only if the comparison determines that it is authorized. This would also be obvious to a person of ordinary skill in the art, as the purpose of comparing the call to the security policy is to determine whether or not it is authorized, so executing the procedure requested by the operating system call before determining whether it is authorized would risk executing unauthorized, potentially malicious operations.

75. Additionally, Jaeger's runtime environment monitor forwards a message to a response engine when the comparison by the runtime environment monitor indicates a violation of one or more of the multiple security rules. In particular, the "authorize" function sends a message "FALSE" as its return value when it determines that a security rule has been violated, indicating that the operation is not authorized.

76. Moreover, as also discussed below for claim element 1.4, it would have been obvious, and well within the skill of a person of ordinary skill in the art, to add additional information to Jaeger's return message. For example, the message could include more information regarding which predicate operation or operations were found to violate rules of the security policy. This would be useful for several reasons, including debugging faulty code, allowing users to increase access rights (e.g., if the operation is desired and the security policy too restrictive), and identifying attacks by malicious code. For example, U.S. Patent No. 5,867,647 to Haigh *et al*. (EX1014), filed February 9, 1996, describes a system in which "compiled program code is allowed to execute and type enforcement violations are logged," allowing the environment of the code to be modified to prevent recurrence of the violations. EX1014 at Abstract. Furthermore, Jaeger's system already generates information identifying an unauthorized predicate operation in the nested FOR loop of its "authorize" function (as the position in the loop

corresponds to a particular "arg" and "op"), so including this information in the message sent to the response engine would be a straightforward process for a person of ordinary skill, and such a person would have a reasonable expectation of success in doing so.

Claim 1.4: a response engine for compiling each rule violation indicated in the messages forwarded by the runtime environment monitor, for blocking execution of operating system calls that are forbidden according to the security policy when execution of the operating system calls would result in a violation of a predetermined combination of multiple security rules of the predetermined security policy and for allowing execution of operating system calls that are permitted according to the security policy;

77. The '520 patent describes a "response engine" as software that responds to a security violation detected by the comparator by taking an appropriate action:

> If the OS request does not violate a security rule 330, then the response engine 318 in step 730 instructs the operating system 260 via the recognizing probe 310-316 to resume operation of the OS request. Otherwise, if the OS request violates a security rule 330, then the response engine 318 in step 730 manages the suspicious Downloadable by performing the appropriate predetermined responsive actions as described with reference to FIGS. 5 and 6.

EX1008 at 6:41-51. The '520 patent describes a variety of "appropriate predetermined responsive action[s]" that could be taken in response to a rule violation such as "stopping execution of a suspicious Downloadable." EX1008 at 6:6-13. Accordingly, in at least one example, a response engine according to the

’755 patent executes a requested procedure if the comparator determines no violation and does not execute the procedure if the comparator determines that there is a violation of a security rule.

78. Jaeger teaches a response engine for compiling each rule violation indicated in the messages forwarded by the runtime environment monitor. As discussed above in the previous claim element, this function is performed by the “operation” function. *See* EX1005 at Fig. 8 In the specific example shown, any violation of the security rules is received as a message from the authorize function as a FALSE return value. For each rule violation detected, the return value of FALSE is recorded and used to evaluate the IF statement in the line “If (authorize args arg_types arg_ops).” When the IF statement evaluates a FALSE value, it skips the THEN line that follows, thus avoiding execution of the requested operation.

79. Although claim 1 neither requires more than one violation to be included in each message or to be compiled with a second violation, nor requires any additional information (e.g., characterizing the violation) to be included in the message, either feature would have been obvious to a person of ordinary skill in the art. As discussed in the previous claim element, it would have been obvious to include additional information regarding which predicate operation or operations were found to violate rules of the security policy. Reasons for doing so would have included debugging faulty code, allowing users to increase access rights (e.g., if

the operation is desired and the security policy too restrictive), and identifying attacks by malicious code. Moreover, this information is already generated at least implicitly in the "authorize" function, which identifies each violation in terms of the "op" and argument that causes the violation, so conveying it in a message would have been simple to include. Furthermore, compiling (e.g., processing or recording) such information upon receipt would have been a necessary addition, as failing to do so would defeat the purpose of transmitting it in the first place.

80. Additionally, Jaeger's system blocks execution of operating system calls that are forbidden according to the security policy when execution of the operating system calls would result in a violation of a predetermined combination of multiple security rules of the predetermined security policy. Jaeger's "operation" function blocks execution of any operation that would violate a security policy because the operation is only executed if the "authorize" function returns the value TRUE. And as discussed above, the "authorize" function returns TRUE if the operation comports with the security policy, and FALSE if it violates the security policy. Furthermore, Jaeger's security policy is a predetermined combination of security rules. For example, Jaeger states that the "access rights available to the content are an intersection of: (1) the rights the downloading principal grants to the remote principal to execute the application; (2) the rights that the downloading principal grants to the application developer for the

application; and (3) the rights that the application grants." EX1005 at 0007. As an intersection of three different sets of rights, the policy enforced by Jaeger is a predetermined combination of multiple security rules of the predetermined security policy, as recited in claim 1.

81. In addition, Jaeger's system allows execution of operating system calls that are permitted according to the security policy. This occurs in the "operation" function, as demonstrated by the following lines of pseudocode:

```
If (authorize args arg_types arg_ops)
        then results = execute(procedure, args)
```

EX1005 at Fig. 8. The first line determines whether the set of predicate operations of the request (and of its arguments) are authorized (and thus allowed by the security policy). The second line executes the requested operation (procedure) if allowed. As Jaeger describes it, "[o]nce authorized, the browser can execute the procedure." *Id*. at 0015.

Claim 1.5: a downloadable engine for intercepting a request message being issued by a downloadable to an operating system, wherein the request message includes an extension call;

82. Jaeger includes a downloadable engine for intercepting a request message being issued by a downloadable to an operating system. Jaeger states that "[o]bviously, the access requests made directly by content must be controlled." EX1005 at 0005. Jaeger describes a browser that assigns downloaded content to application-specific interpreters for execution based on the content type. *Id*. at

0007-08, Figs. 4, 5. Jaeger's system therefore comprises software to manage and execute downloaded content, corresponding to the meaning of a downloadable engine in the '755 patent.

83. Jaeger describes how execution is handled by the browser and interpreters:

> After the browser authenticates the content and sends the content to be run in the appropriate application-specific interpreter as described above, the content is assigned to content interpreter based on the identity provided by authentication. This content interpreter has access rights consistent with the identity of the remote principal. Any controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed.

Id. at 0008. Jaeger also describes the interpreters as specifically handling access requests. For example, in the case of a numerical analysis package, the "application-specific interpreter requests execution of the logical numerical analysis package object, and the browser maps the logical object to the actual system object." EX1005 at 0014. Thus, when downloaded content requests execution of a resource, Jaeger's downloadable engine intercepts the request message and handles execution itself.

84. Furthermore, Jaeger's request message can include an extension call. As discussed above, in the '755 patent, the term "extension call" refers to "a

request corresponding to a class of program objects." Jaeger explicitly suggests including components in the system to handle requests corresponding to classes of program objects. For example, Jaeger states that "if the software should be further restricted or if we want to be sure that the software can be executed successfully, then we suggest the creation of classes for software to represent this information." EX1005 at 0016. Furthermore, Jaeger describes access control relationships in its system, including specifying that "principal groups can execute operations on objects that belong to classes (class operations)" EX1005 at Fig. 6; *see also id*. at 0007-08 (defining "class operations" for objects in its system).

85. Insofar as the "extension call" of claim 1 requires that the "extension" be a specific extension type, such as an extension for a Java class, such a feature would have been obvious to a person of ordinary skill in the art. The Java language and associated Java applets were well-known in 1996; for example, Jaeger lists "Current interpreters for executing downloaded content, such as Java-enabled Netscape [and] Java's appletviewer" and describes that "Java appletviewer permits some access rights to be granted to content." EX1005 at 0002-03; *see also id*. at Fig. 2, 0004-05. When requesting access to outside resources such as network or file systems, a Java applet would generate an extension call corresponding to an appropriate class of program object.

86. As would have been well known to a person of ordinary skill in the art in 1996, Java-enabled Netscape was a browser that included the ability to execute Java using the Java virtual machine. For example, U.S. Patent No. 5,754,830, filed April 1, 1996, describes "JAVA-capable NETSCAPE NAVIGATOR" as having a "JAVA virtual machine within the NETSCAPE NAVIGATOR web browser". EX1022 at 5:17-29. I note that the Java virtual machine is given as an example of a downloadable engine in the '520 patent. *See* EX1008 at 3:42-46.

87. It would also have been obvious, as discussed below, to use a request broker as disclosed in CORBA to enable the authorization components of the system of Jaeger to communicate with a downloadable engine, such as the Java virtual machine of the Java-enabled Netscape browser disclosed by Jaeger. Furthermore, the application of class-based access rights, as suggested by Jaeger, to allow the interpreters of Jaeger to process Java extension calls, or extension calls in other languages, would be a routine matter for a person of ordinary skill in the art in 1996.

Claim 1.6: a request broker for receiving a notification message from the downloadable engine regarding the extension call;

88. Claim 1.6 describes a request broker for receiving a notification message from the downloadable engine regarding the extension call. Claim 1.8 recites an event router for receiving the notification message from the request broker. The event router then forwards messages it receives (including the

notification message) to the runtime event monitor. The request broker is not described as taking any action other than receiving the notification message. Accordingly, it appears the request broker can be a simple connection that forwards a message from the downloadable engine to the event router. This matches the description of the request broker in the '520 patent which describes "request broker 306, which forwards the message to the event router 308." EX1008 at 5:39-41.

89. Jaeger discloses such a request broker for receiving a notification message from the downloadable engine regarding the extension call. In particular, in discussing certain operations, Jaeger discloses that the "operations are implemented by a call to authorize which checks the content access rights prior to calling the actual operation. Since these commands are only available in the browser and are protected by the authorization operation, only authorized accesses to system objects are not possible." EX1005 at 0016. Jaeger also discusses transmitting such calls in the context of class-based access rights:

> If the software should be further restricted or if we want to be sure that the software can be executed successfully, then we suggest the creation of classes for software to represent this information. Software classes contain strongly-typed constructors for executing the software. Also, access control predicates can be assigned to the arguments in the call.

Id. As illustrated in Fig. 5, Jaeger's system receives request calls from content and sends calls between the browser and application-specific interpreters to authorize and execute the requested operations. A person of ordinary skill would therefore understand that Jaeger's system would include a request broker

90. Furthermore, it would have been obvious to include a request broker in Jaeger's system to handle calls from class extensions, as the use of a request broker to establish communication between objects in a distributed computing architecture was well known. For example, standards for the use of request brokers in such architectures are set out in "The Common Object Request Broker: Architecture and Specification" ("CORBA", EX1011). CORBA illustrates the structure of an object request broker ("ORB") in Fig. 2, which "shows a request being sent by a client to an object implementation." EX1011 at 30. CORBA describes that "ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request." *Id*. CORBA describes that the architecture is "structured to allow integration of a wide variety of object systems." *Id*. at 29. For example, the request brokers of CORBA can receive and forward messages for objects in an object oriented code, including objects organized by class. *See, e.g.*, EX1011 at 152 (describing "a Basic Object Adapter" that "should be available in every ORB," and which can "provide the

Object Implementation," such as a "per-class server"). Accordingly, Jaeger in view of CORBA discloses a request broker for receiving a notification message from the downloadable engine regarding the extension call.

91. It would have been obvious to use a request broker, as disclosed in CORBA, in the system of Jaeger. Jaeger discloses a distributed system architecture and suggests using "classes of objects" to control access rights. *E.g.*, EX1005 at 0002, 0009, 0016. Jaeger further requires requests to be forwarded from a downloadable to application-independent and application-specific interpreters. EX1005 at 0007-08. CORBA discloses that request brokers can be used in to "provide[] interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnect[] multiple object systems." EX1011 at 15. CORBA further explains that its "object model is an example of a classical object model, where a client sends a message to an object." EX1011 at 20. This matches the system of Jaeger, where content messages are sent to objects associated with application-independent and -specific interpreters for execution. Using the request broker of CORBA to pass notification messages regarding an extension call in Jaeger's system would be obvious, as it would represent a combination of prior art elements according to known methods to yield predictable results.

Claim 1.7: a file system probe and a network system probe each being associated with an operating system function for receiving the request

message from the downloadable engine and intercepting an operating system call being issued by the downloadable to an operating system and associated with the operating system function;

92. Jaeger discloses a file system probe and a network system probe each being associated with an operating system function for receiving the request message from the downloadable engine and intercepting an operating system call being issued by the downloadable to an operating system and associated with the operating system function. As discussed above in claim 1.2, Jaeger discloses operating system probes.

93. For example, Jaeger discloses "Trusted, application-independent interpreters (which we will refer to as browsers from here on) control access to system objects by the application-specific interpreters and downloaded content. An application-specific interpreter controls access to application objects and specifies the access rights of downloaded content within its limited domain." EX1005 at 0007. Jaeger describes the system as containing multiple interpreters (*e.g.*, the browser and each of the various application-specific interpreters), each including one or more probes, so that each different type of operation can be authorized and executed: "the browser determines which interpreter to execute the content. If the type refers to a known application-specific interpreter and the remote principal has the rights to execute content in that interpreter, then the content is run in that interpreter." *Id.* at 0007. Jaeger describes the respective interpreters as being used

to monitor different subsystems, such as "Application Objects," "File System," "Processes," "Environment," and "Network Services." *Id.* at 0005. The probes associated with the "File System" and "Network Services" constitute "a file system probe and a network system probe each being associated with an operating system function."

94. Jaeger also specifies that the probes intercept operating system calls being issued by the downloadable to an operating system and associated with the operating system function:

> Content is executed as shown in Figure 5. . . . Any controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed. Operations are executed in the interpreter trusted with the operation.

EX1005 at 0008. If the interpreter authorizes controlled operations run by content, and then executes those requests, the interpreter's probes necessarily handle requests during runtime.

95. EX1005 at 0008. The protocol for content execution in Jaeger's system is shown in Fig. 5:

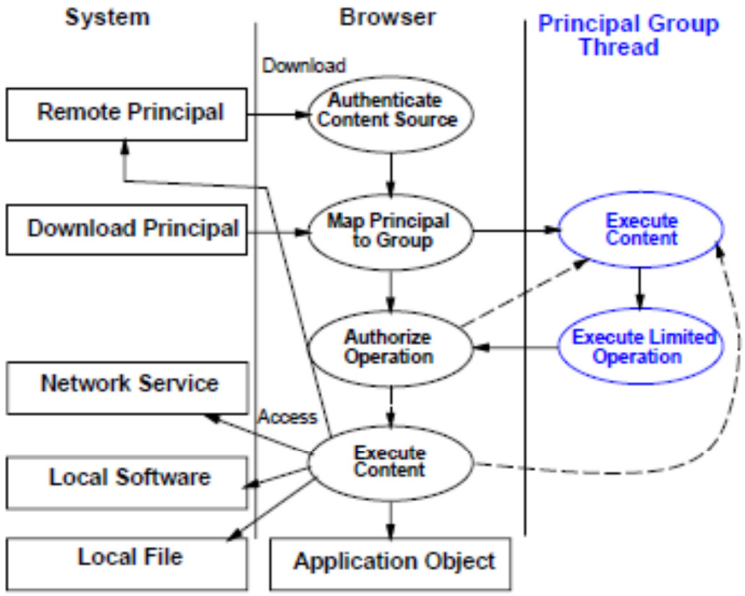


Figure 5: **Content Execution Protocol**: (1) Remote principal downloads content to browser; (2) Browser assigns content to principal group in application-specific interpreter (not shown); (3) content executes a limited operation which cause authorization in the browser; (4) if authorized the browser access system objects on behalf of content.

Fig. 5 clearly shows requests from the downloaded content being intercepted, handled by the browser and associated application-specific interpreters, as appropriate, then executed to access operating system objects including files and network services.

96. In particular, Jaeger's system could receive an extension call from an object corresponding to a class (such as a Java object), where that call sought access to a file or network system. *See, e.g.*, EX1005 at 0003 (discussing "file system I/O and communicating with third parties" being allowed for "Java

appletviewer."), 0005 (discussing granting to Java applets "the right to communicate to … IP addresses" and "read and write access rights to files"). Such a request would be handled by sending a message to the appropriate network or file system probe of Jaeger's browser, as shown in Fig. 5. Accordingly, Jaeger discloses each element of claim 1.7.

Claim 1.8: an event router for receiving the notification message from the request broker regarding the extension call and an event message from one of the file system probe and the network system probe regarding the operating system call;

97. Jaeger discloses an event router for receiving the notification message from the request broker regarding the extension call and an event message from one of the file system probe and the network system probe regarding the operating system call. I note that the event message is "from one of the file system probe and the network system probe regarding the operating system call." As discussed in element 1.7, in Jaeger's system, each of these probes intercepts operating system calls from its respective system. The probes also send event messages requesting authorization of the operations, as illustrated in Fig. 5 and discussed below.

98. Jaeger discloses that the application independent interpreter (browser) receives requests from downloadables and assigns a corresponding interpreter to authorize their execution. *See* EX1005 at Figs. 3, 5. Jaeger describes the function of the browser in the architecture as follows:

> Lastly, the architecture provides support for enforcing the access rights specified above while executing content and any external software and network services used by the content. The browser: (1) authorizes content operations; (2) provides safe implementations of these operations; (3) enables controlled execution of external software and network services.

EX1005 at 0015. As the browser receives each operation request, it would in particular receive both the notification message and the event message discussed above. Furthermore, it forwards the message to the appropriate interpreter, comprising a runtime event monitor, as required by claim 1.9, below. *See* EX1005 at Fig. 4 (the browser "authenticates the remote principal and finds the appropriate interpreter to execute the content. If the remote principal is authorized to execute content in that interpreter, then the content is executed"). Indeed, as each message requesting authorization must be received by the authorization subroutines (*see* EX1005 at Fig. 5), it would have been obvious to include an event router in Jaeger's system. Accordingly, it would have been obvious for the browser of Jaeger's system to comprise an event router.

Claim 1.9: the runtime environment monitor for receiving the notification message and the event message from the event router and comparing the extension call and the operating system call against a predetermined security policy before allowing the operating system to process the extension call and the operating system call; and

99. As discussed above in claim 1.3, Jaeger teaches a runtime environment monitor for comparing the operating system call against a

predetermined security policy including multiple security rules to determine if execution of the operating system call violates one or more of the multiple security rules. *See, e.g.*, EX1005 at 0012, 0014, Figs. 8 and 8. Likewise does the runtime event monitor of Jaeger compare the extension call against the predetermined security policy, as Jaeger states that every operation must be authorized (confirmed to adhere to the security policy) before execution. *See* EX1005 at 0005 ("Operations on application objects must be authorized to ensure that the principal is permitted to modify the object"), 0008 ("Any controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed."), 0014 ("a call to the authorization procedure (described below) is added after the access control of each controlled operation"); see also id. at Figs. 8, 9 (showing "authorize" procedure of Jaeger's runtime environment monitor called by "operation" routine). As discussed above in claim 1.8, the event router of Jaeger's browser sends each request message to the appropriate interpreter, which comprises a runtime event monitor. Accordingly, Jaeger in view of CORBA discloses claim 1.9.

Claim 1.10: the response engine for receiving a violation message from the runtime environment monitor when one of the extension call and the operating system call violate one or more rules of the predetermined security policy and blocking extension calls and operating system calls that are forbidden according to the predetermined security policy, and for allowing extension calls and operating system calls that are permitted according to the predetermined security policy.

100. As discussed above in claim 1.4, the response engine of Jaeger receives a violation message from the runtime environment monitor whenever a security rule of the security policy is violated (e.g., as illustrated in Figs. 8 and 9 of Jaeger). As further discussed in claim 1.4, Jaeger's response engine allows operations that are permitted and blocks operations that are forbidden, as indicated by the code "If (authorize args arg_types arg_ops) then results = execute(procedure, args)." EX1005 at Fig. 8. And as discussed above in claim 1.9, this applies to every call, including both operating system calls and extension calls: "**Any** controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed." *Id*. at 0008 (emphasis added).

101. Jaeger's system would thus compare extension calls to the security policy just as it would compare operating system calls or other requests from the downloadable to the security policy. Therefore, Jaeger discloses "the response engine for receiving a violation message from the runtime environment monitor when one of the extension call and the operating system call violate one or more

rules of the predetermined security policy and blocking extension calls and operating system calls that are forbidden according to the predetermined security policy, and for allowing extension calls and operating system calls that are permitted according to the predetermined security policy," as recited in claim 1.10.

102. As discussed above, Jaeger in view of CORBA discloses each and every element of claim 1, and a person of ordinary skill in the art would have been motivated to combine the references in the manner described in claim 1, with a reasonable expectation of success in doing so. Accordingly, claim 1 would have been obvious to a person of ordinary skill in the art.

Claim 2: The system of claim 1, wherein the downloadable is one of a Java component, an ActiveX control, executable code, or interpretable code.

103. Claim 2 depends from claim 1, and specifies that the Downloadable is a Java component, an ActiveX control, executable code, or interpretable code. As discussed below, Jaeger in view of CORBA renders this claim obvious.

104. As discussed above, Jaeger in view of CORBA renders claim 1 obvious. Jaeger also discloses wherein the Downloadable is executable code. Indeed, the title of Jaeger is "Building Systems That Flexibly Control Downloaded Executable Content," and Fig. 3 illustrates such a system. *See also* EX1005 at Fig. 5 (illustrating steps including "Download," "Authenticate Content Source," "Authorize Operation," and "Execute Content"). Similarly, it would have been obvious for the Downloadable to be a Java component, an ActiveX control, or

interpretable code, as each of these was a well-known type of Downloadable in 1996. *See, e.g.*, Ex1005 at 0002-03 ("Current interpreters for executing downloaded content, such as Java-enabled Netscape, Java's appletviewer [10], and Tcl's safe interpreter"). Accordingly, claim 2 would have been obvious in view of Jaeger and CORBA.

Claim 3, preamble: A system for reviewing an operating system call issued by a downloadable, comprising:

105. Claim 3 is an independent claim with similar scope to claim 1. Claim 3 differs from claim 1 primarily in that claim 3 does not include claim elements corresponding to claim elements 1.2, 1.3, or 1.4 (discussing operating system probe and interactions between the probe, the runtime environment monitor, and the response engine, including compiling one or more messages), nor does it require an event router, as recited in claim 1.8. Accordingly, I apply substantially similar analysis to claim 3 as I do to claim 1.

106. Jaeger discloses a system for reviewing an operating system call issued by a downloadable. Jaeger describes "Systems That Flexibly Control Downloaded Executable Content." EX1005, Title. Jaeger describes the overall functions of the system as follows: "(1) authenticate the source of the content; (2) determine the least privilege access rights for the content given its source; and (3) enforce those access rights throughout the execution of the content." *Id*. at 0004. Jaeger describes the use of a "conventional protection model," in which "principals

(e.g., users, groups, services, etc.) execute processes that perform operations (e.g., read, write, etc.) on objects (e.g., files, devices, etc.). The permissions of a principal to perform operations on objects are called the access rights of the principal. We call a set of access rights an access control domain or simply domain." EX1005 at 0003. Jaeger details the types of objects for which "access . . . needs to be controlled," including applications, file system objects, processes, network services, and the environment of each process. *Id.* at 0005. Accessing these object types involves calls to an operating system. Indeed, Jaeger specifies that the system identifies "I/O commands . . . to control access to system objects" and that the system includes an "operating system [that] has an unmodified trusted computing base protects process domains. and provides authentication of principals." *Id.* at 0006-07. Accordingly, Jaeger discloses a system for reviewing an operating system call issued by a downloadable.

Claim 3.1: at least one processor for accessing elements stored in at least one memory associated with the at least one processor and for executing instructions associated with the elements, the elements including:

107. Jaeger's system includes a processor for accessing elements stored in at least one memory associated with the at least one processor and for executing instructions associated with the elements. Jaeger discloses "an architecture that flexibly controls the access rights of downloaded content." EX1005 at 0002. Jaeger describes this architecture as employing an "operating system [that] has an

unmodified trusted computing base." *Id.* at 0006; *see also id.* at Fig. 2 (showing a system running on Network Computing Environments). A person of ordinary skill in the art reading Jaeger would understand that the computing base would include a processor and associated memory, for executing instructions. Jaeger further discloses that

> In general, access to the following types of objects needs to be controlled:
>
> - Application Objects: A process can read and write objects in its memory
> - File System: A process can perform read, write, and execute operations
> - Applications: A process can execute another application by creating a process for that application

Id. at 0005.

108. Accordingly, Jaeger teaches that the processor (running Jaeger's "process") is capable of accessing elements stored in the at least one memory (such as the "objects in its memory"), and that the processor executes the associated instructions. Furthermore, as Jaeger's architecture is intended to run on a computer, it would have been obvious to a person of ordinary skill to include a processor for executing instructions stored in associated memory, as such elements represent two of the essential building blocks for all of modern computer science. For similar reasons, a person of ordinary skill would have a reasonable expectation

of success in making use of such elements. The system disclosed by Jaeger would be no exception; indeed, Jaeger states that the system's "model is influenced most strongly by the access control models of Hydra," which is a "kernel of a multiprocessor system." EX1005 at 0010, 0019.

Claim 3.2: a downloadable engine for intercepting a request message being issued by a downloadable to an operating system, wherein the request message includes an extension call;

109. Jaeger includes a downloadable engine for intercepting a request message being issued by a downloadable to an operating system. Jaeger states that "[o]bviously, the access requests made directly by content must be controlled." EX1005 at 0005. Jaeger describes a browser that assigns downloaded content to application-specific interpreters for execution based on the content type. *Id*. at 0007-08, Figs. 4, 5. Jaeger's system therefore comprises software to manage and execute downloaded content, corresponding to the meaning of a downloadable engine in the '755 patent.

110. Jaeger describes how execution is handled by the browser and interpreters:

> After the browser authenticates the content and sends the content to be run in the appropriate application-specific interpreter as described above, the content is assigned to content interpreter based on the identity provided by authentication. This content interpreter has access rights consistent with the identity of the

> remote principal. Any controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed.

Id. at 0008.

111. Jaeger also describes the interpreters as specifically handling access requests. For example, in the case of a numerical analysis package, the "application-specific interpreter requests execution of the logical numerical analysis package object, and the browser maps the logical object to the actual system object." EX1005 at 0014. Thus, when downloaded content requests execution of a resource, Jaeger's downloadable engine intercepts the request message and handles execution itself.

112. Furthermore, Jaeger's request message can include an extension call. As discussed above, in the '755 patent, the term "extension call" refers to "a request corresponding to a class of program objects." Jaeger explicitly suggests including components in the system to handle requests corresponding to classes of program objects. For example, Jaeger states that if "the software should be further restricted or if we want to be sure that the software can be executed successfully, then we suggest the creation of classes for software to represent this information." Furthermore, Jaeger describes access control relationships in its system, including specifying that "principal groups can execute operations on objects that belong to

classes (class operations)" EX1005 at Fig. 6; *see also id*. at 7-8 (defining "class operations" for objects in its system).

113. Insofar as the "extension call" of claim 1 requires that the "extension" be a specific extension type, such as an extension for a Java class, such a feature would have been obvious to a person of ordinary skill in the art. The Java language and associated Java applets were well-known in 1996; for example, Jaeger lists "Current interpreters for executing downloaded content, such as Java-enabled Netscape [and] Java's appletviewer" and describes that "Java appletviewer permits some access rights to be granted to content." EX1005 at 0002-03; *see also id*. at Fig. 2, 0004-05. When requesting access to outside resources such as network or file systems, a Java applet would generate an extension call corresponding to an appropriate class of program object.

114. As would have been well known to a person of ordinary skill in the art in 1996, Java-enabled Netscape was a browser that included the ability to execute Java using the Java virtual machine. For example, U.S. Patent No. 5,754,830, filed April 1, 1996, describes "JAVA-capable NETSCAPE NAVIGATOR" as having a "JAVA virtual machine within the NETSCAPE NAVIGATOR web browser". EX1022 at 5:17-29. I note that the Java virtual machine is given as an example of a downloadable engine in the '520 patent. *See* EX1008 at 3:42-46.

115. It would also have been obvious, as discussed below, to use a request broker as disclosed in CORBA to enable the authorization components of the system of Jaeger to communicate with a downloadable engine, such as the Java virtual machine of the Java-enabled Netscape browser disclosed by Jaeger. Furthermore, the application of class-based access rights, as suggested by Jaeger, to allow the interpreters of Jaeger to process Java extension calls, or extension calls in other languages, would be a routine matter for a person of ordinary skill in the art in 1996.

Claim 3.3: a request broker for receiving a notification message from the downloadable engine regarding the extension call;

116. Claim 3.3 describes a request broker for receiving a notification message from the downloadable engine regarding the extension call. The request broker is not described as taking any action other than receiving the notification message. The notification message is also received by the runtime event monitor (*see below,* claim 3.6), although claim 3 does not specify whether the request broker forwards the notification message, receives it from the runtime event monitor, or receives it in parallel with the runtime event monitor from the downloadable engine. At any rate, it appears the request broker can be a simple connection that forwards a message from the downloadable engine to the event router. This matches the description of the request broker in the '520 patent which

describes "request broker 306, which forwards the message to the event router 308." EX1008 at 5:39-41.

117. Jaeger discloses such a request broker for receiving a notification message from the downloadable engine regarding the extension call. In particular, in discussing certain operations, Jaeger discloses that the "operations are implemented by a call to authorize which checks the content access rights prior to calling the actual operation. Since these commands are only available in the browser and are protected by the authorization operation, only authorized accesses to system objects are not possible." EX1005 at 0016. Jaeger also discusses transmitting such calls in the context of class-based access rights:

> If the software should be further restricted or if we want to be sure that the software can be executed successfully, then we suggest the creation of classes for software to represent this information. Software classes contain strongly-typed constructors for executing the software. Also, access control predicates can be assigned to the arguments in the call.

Id.

118. As illustrated in Fig. 5, Jaeger's system receives request calls from content and sends calls between the browser and application-specific interpreters to authorize and execute the requested operations. A person of ordinary skill would therefore understand that Jaeger's system would include a request broker

119. Furthermore, it would have been obvious to include a request broker in Jaeger's system to handle calls from class extensions, as the use of a request broker to establish communication between objects in a distributed computing architecture was well known. For example, standards for the use of request brokers in such architectures are set out in "The Common Object Request Broker: Architecture and Specification" ("CORBA", EX1011). CORBA illustrates the structure of an object request broker ("ORB") in Fig. 2, which "shows a request being sent by a client to an object implementation." EX1011 at 30. CORBA describes that "ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request." *Id*. CORBA describes that the architecture is "structured to allow integration of a wide variety of object systems." *Id*. at 29. For example, the request brokers of CORBA can receive and forward messages for objects in an object oriented code, including objects organized by class. *See, e.g.*, EX1011 at 152 (describing "a Basic Object Adapter" that "should be available in every ORB," and which can "provide the Object Implementation," such as a "per-class server"). Accordingly, Jaeger in view of CORBA discloses a request broker for receiving a notification message from the downloadable engine regarding the extension call.

120. It would have been obvious to use a request broker, as disclosed in CORBA, in the system of Jaeger. Jaeger discloses a distributed system architecture and suggests using "classes of objects" to control access rights. *E.g.*, EX1005 at 0002, 0009, 0016. Jaeger further requires requests to be forwarded from a downloadable to application-independent and application-specific interpreters. EX1005 at 0007-08. CORBA discloses that request brokers can be used in to "provide[] interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnect[] multiple object systems." EX1011 at 15. CORBA further explains that its "object model is an example of a classical object model, where a client sends a message to an object." EX1011 at 20. This matches the system of Jaeger, where content messages are sent to objects associated with application-independent and -specific interpreters for execution. Using the request broker of CORBA to pass notification messages regarding an extension call in Jaeger's system would be obvious, as it would represent a combination of prior art elements according to known methods to yield predictable results.

Claim 3.4: a file system probe and a network system probe each being associated with an operating system function for receiving the request message from the downloadable engine, intercepting an operating system call being issued by the downloadable to an operating system and associated with the operating system function;

121. Jaeger discloses a file system probe and a network system probe each being associated with an operating system function for receiving the request message from the downloadable engine and intercepting an operating system call being issued by the downloadable to an operating system and associated with the operating system function. As discussed above (*see, e.g.*, claim 1.2 and 1.7), Jaeger discloses operating system probes.

122. For example, For example, Jaeger discloses "Trusted, application-independent interpreters (which we will refer to as browsers from here on) control access to system objects by the application-specific interpreters and downloaded content. An application-specific interpreter controls access to application objects and specifies the access rights of downloaded content within its limited domain." EX1005 at 0007. Jaeger describes the system as containing multiple interpreters (*e.g.*, the browser and each of the various application-specific interpreters), each including one or more probes, so that each different type of operation can be authorized and executed: "the browser determines which interpreter to execute the content. If the type refers to a known application-specific interpreter and the remote principal has the rights to execute content in that interpreter, then the

content is run in that interpreter." *Id.* at 0007. Jaeger describes the respective interpreters as being used to monitor different subsystems, such as "Application Objects," "File System," "Processes," "Environment," and "Network Services." *Id.* at 0005. The probes associated with the "File System" and "Network Services" constitute "a file system probe and a network system probe each being associated with an operating system function."

123. Jaeger also specifies that the probes intercept operating system calls being issued by the downloadable to an operating system and associated with the operating system function:

> Content is executed as shown in Figure 5. . . . Any controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed. Operations are executed in the interpreter trusted with the operation.

EX1005 at 0008. If the interpreter authorizes controlled operations run by content, and then executes those requests, the interpreter's probes necessarily handle requests during runtime.

124. The protocol for content execution in Jaeger's system is shown in Fig. 5:

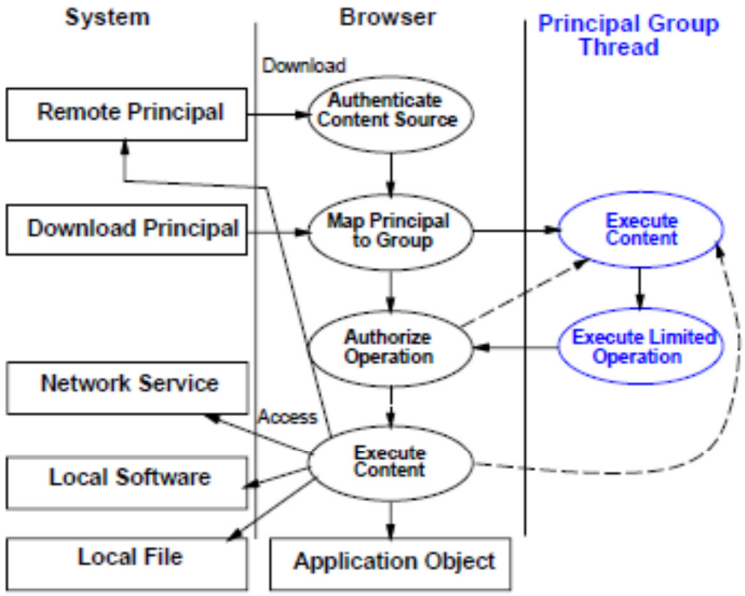


Figure 5: **Content Execution Protocol**: (1) Remote principal downloads content to browser; (2) Browser assigns content to principal group in application-specific interpreter (not shown); (3) content executes a limited operation which cause authorization in the browser; (4) if authorized the browser access system objects on behalf of content.

Fig. 5 clearly shows requests from the downloaded content being intercepted, handled by the browser and associated application-specific interpreters, as appropriate, then executed to access operating system objects including files and network services.

125. In particular, Jaeger's system could receive an extension call from an object corresponding to a class (such as a Java object), where that call sought access to a file or network system. *See, e.g.*, EX1005 at 0003 (discussing "file system I/O and communicating with third parties" being allowed for "Java

appletviewer."), 0005 (discussing granting to Java applets "the right to communicate to … IP addresses" and "read and write access rights to files"). Such a request would be handled by sending a message to the appropriate network or file system probe of Jaeger's browser, as shown in Fig. 5. Accordingly, Jaeger discloses each element of claim 3.4.

Claim 3.5: and providing an event message regarding the operating system call

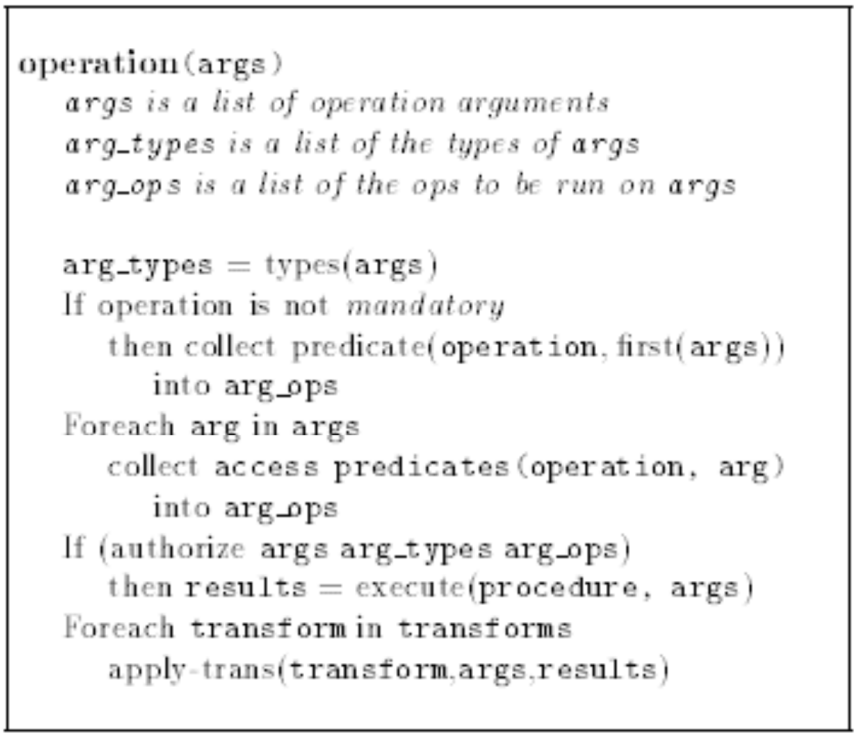
126. Jaeger discloses the file system probe and network system probe for providing an event message regarding the operating system call. As discussed in element 3.4, in Jaeger's system, each of these probes intercepts operating system calls related to its respective system. The probes also send event messages requesting authorization of the operations, as illustrated in Fig. 5 and discussed above in claim element 1.8. Accordingly, Jaeger discloses the file and network system probes providing an event message regarding the operating system call.

Claim 3.6: a runtime environment monitor for receiving the notification message and the event message and comparing the extension call and the operating system call against a predetermined security policy before allowing the operating system to process the extension call and the operating system call; and

127. As discussed above in claims 1.3 and 1.9, Jaeger teaches a runtime environment monitor for comparing the extension call and the operating system call against a predetermined security policy including multiple security rules to determine if execution of the operating system call violates one or more of the

multiple security rules. Jaeger defines security policies according to "access rights." These access rights of a given correspond to the intersection of access rights the different system components being used. *See* EX1005 at 0012 ("The access rights of a remote principals content in interpreter t active at states are the intersection of the rights of interpreter t and the rights of the principal group to which the remote principal is assigned in interpreter t at state s.") These rights are specified in advance because, as Jaeger describes, "we want delegation to be implicit relative to some application rather than require the downloading principal to specify access rights at runtime." *Id*. at 0014.

128. The runtime event monitor of Jaeger compares both the operating system call and the extension call against the predetermined security policy, as Jaeger states that every operation must be authorized (confirmed to adhere to the security policy) before execution. *See* EX1005 at 0005 ("Operations on application objects must be authorized to ensure that the principal is permitted to modify the object"), 0008 ("Any controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed."), 0014 ("a call to the authorization procedure (described below) is added after the access control of each controlled operation").



```
operation(args)
   args is a list of operation arguments
   arg_types is a list of the types of args
   arg_ops is a list of the ops to be run on args

   arg_types = types(args)
   If operation is not mandatory
      then collect predicate(operation, first(args))
         into arg_ops
   Foreach arg in args
      collect access predicates(operation, arg)
         into arg_ops
   If (authorize args arg_types arg_ops)
      then results = execute(procedure, args)
   Foreach transform in transforms
      apply-trans(transform,args,results)
```

Figure 8: Generated operation with authorization and transformation calls

129. Jaeger provides pseudocode in Figs. 8 and 9 illustrating how a requested operation can be authorized and subsequently executed subject to that authorization. Fig. 8 (above) illustrates a process for handling an operation. A request for an operation is received, including one or more arguments, "args." The types of each argument and the predicate operations ("ops") required for the requested operation each argument thereof are determined and stored as "arg_types" and "arg_ops," respectively. The operation process is subsequently suspended in the line reading "If (authorize args arg_types arg_ops)." This line calls the "authorize" procedure, which is illustrated in Fig. 9 (below). The

"authorize" procedure returns a value of TRUE (FALSE) if the operation is permitted (forbidden) by the security policy. Accordingly, Jaeger's system compares an operating system call to a predetermined security policy and determines whether or not the policy is violated.

```
authorize(args arg_types arg_ops)
   For arg in args, arg_type in arg_types,
      ops in arg_ops
      Find object group for arg and arg_type
      For op in ops
         If domain rights grant op
            AND no exception precludes op on arg
            then continue
            else return FALSE
   return TRUE
```

Figure 9: Authorization procedure

130. Furthermore, Jaeger's system performs the comparison before allowing the operating system to process the operating system call. As shown in Fig. 8 of Jaeger, the "operation" function does not execute the procedure requested by the call until after evaluating whether it is authorized. Indeed, the procedure is executed only if the comparison determines that it is authorized. This would also be obvious to a person of ordinary skill in the art, as the purpose of comparing the call to the security policy is to determine whether or not it is authorized, so

executing the procedure requested by the operating system call before determining whether it is authorized would risk executing unauthorized, potentially malicious operations.

131. Additionally, Jaeger's runtime environment monitor forwards a message to a response engine when the comparison by the runtime environment monitor indicates a violation of one or more of the multiple security rules. In particular, the "authorize" function sends a message "FALSE" as its return value when it determines that a security rule has been violated, indicating that the operation is not authorized.

132. Accordingly, Jaeger in view of CORBA discloses claim 1.9.

Claim 3.7: a response engine for receiving a violation message from the runtime environment monitor when one of the extension call and the operating system call violate one or more rules of the predetermined security policy and blocking extension calls and operating system calls that are forbidden according to the predetermined security policy, and for allowing extension calls and operating system calls that are permitted according to the predetermined security policy.

133. As discussed above in claims 1.4 and 1.10, Jaeger teaches a response engine for receiving a violation message from the runtime environment monitor when one of the extension call and the operating system call violate one or more rules of the predetermined security policy. For example, this function is performed by Jaeger's "operation" function illustrated in Fig. 8.

134. In the specific example shown, any violation of the security rules is received as a message from the authorize function as a FALSE return value. For each rule violation detected, the return value of FALSE is recorded and used to evaluate the IF statement in the line "If (authorize args arg_types arg_ops)." When the IF statement evaluates a FALSE value, it skips the THEN line that follows, thus avoiding execution of the requested (forbidden) operation. On the other hand, if all operations of the request are permitted by the policy, the authorize procedure returns TRUE, and the requested procedure executes in the THEN line of the "operation" function. As Jaeger describes it, "[o]nce authorized, the browser can execute the procedure." *Id.* at 0015.

135. Furthermore, Jaeger's security policy includes one or more security rules. Jaeger describes that "access rights available to the content are an intersection of: (1) the rights the downloading principal grants to the remote principal to execute the application; (2) the rights that the downloading principal grants to the application developer for the application; and (3) the rights that the application grants." EX1005 at 0007. As an intersection of three different sets of rights, the policy enforced by Jaeger is a combination of multiple security rules of the predetermined security policy.

136. Additionally, the response engine of Jaeger performs the above-described functions for every request it processes, including both operating system

calls and extension calls: "**Any** controlled operation run by the content is authorized by the appropriate interpreter and if authorization is granted then the operation is executed." *Id*. at 0008 (emphasis added). Jaeger's system would thus compare extension calls to the security policy just as it would compare operating system calls or other requests from the downloadable to the security policy. Therefore, Jaeger discloses "a response engine for receiving a violation message from the runtime environment monitor when one of the extension call and the operating system call violate one or more rules of the predetermined security policy and blocking extension calls and operating system calls that are forbidden according to the predetermined security policy, and for allowing extension calls and operating system calls that are permitted according to the predetermined security policy." as recited in claim 3.7.

Claim 4: The system of claim 3, wherein the Downloadable is one of a Java component, an ActiveX control, executable code, or interpretable code.

137. Claim 4 depends from claim 3, and specifies that the Downloadable is a Java component, an ActiveX control, executable code, or interpretable code. As discussed below, Jaeger in view of CORBA renders this claim obvious.

138. As discussed above, Jaeger in view of CORBA renders claim 3 obvious. Jaeger also discloses wherein the Downloadable is executable code. Indeed, the title of Jaeger is "Building Systems That Flexibly Control Downloaded Executable Content," and Fig. 3 illustrates such a system. *See also* EX1005 at Fig.

5 (illustrating steps including "Download," "Authenticate Content Source," "Authorize Operation," and "Execute Content"). Similarly, it would have been obvious for the Downloadable to be a Java component, an ActiveX control, or interpretable code, as each of these was a well-known type of Downloadable in 1996. *See, e.g.*, Ex1005 at 0002-03 ("Current interpreters for executing downloaded content, such as Java-enabled Netscape, Java's appletviewer [10], and Tcl's safe interpreter"). Accordingly, claim 4 would have been obvious in view of Jaeger and CORBA.

139. Accordingly, for the reasons discussed above, it is my opinion that claims 1-4 of the '755 patent would have been obvious to a person of ordinary skill in the art in view of Jaeger and CORBA.

IX. GROUND 2: CLAIM 5 IS OBVIOUS OVER JAEGER IN VIEW OF CORBA AND TBAV

140. As explained in detail below, it is my opinion that each and every element of claim 5 of the '755 patent can be found in the prior art, including the references identified below.

141. Claim 5 of the '755 patent is presented below in bold text followed by my analysis of the claim. The analysis below identifies exemplary disclosure of the cited references relative to the corresponding claim elements, and it is not meant to be exhaustive.

Claim 5: The system of claim 3, wherein the response engine stores results of the comparison in an event log.

142. Claim 5 of the '755 patent depends from claim 3. My analysis above in Ground 1 with respect to claim 3 explains how Jaeger in view of CORBA renders obvious each and every element of claim 3.

143. Jaeger discloses detecting unauthorized operation requests based on the comparison of information pertaining to the downloadable and a predetermined security policy, as discussed above in claim 3. A function of Jaeger's system is to prevent a "malicious remote principal" from "obtain[ing] unauthorized access to the downloading principal's resources." EX1005 at 0002. For example, Jaeger contemplates that "a remote principal may be spoofed into integrating malicious content from an attacker to mobile agent which would then give a third party access to the information provided above." *Id.* An attempt to access unauthorized resources, as determined by the "authorize operation" of Jaeger's system, would serve as an indicator that a particular piece of content was malicious, or at least suspicious. For example, Jaeger describes security problems including an "application which could try to perform, either accidentally or maliciously, unauthorized actions." *Id.* at 0004.

144. A person of ordinary skill would understand that the results of Jaeger's comparison would constitute a useful indication of whether or not the content in question was suspicious. Jaeger discloses storing results in a variable.

See Fig. 8 (including the variable "results"). Jaeger also discloses the use of data logging for its interpreters, suggesting that the contents of a file used to define mappings from object groups to system objects "should be based on analysis, *e.g.*, use in system that can log process actions of the interpreter t." EX1005 at 0013. To the extent that Jaeger does not explicitly disclose storing the results of its runtime environment monitor's comparison in a log file, a person of ordinary skill would have found it obvious to do so, as the storage of such information in a log file was a well-established, routine practice when dealing with potentially malicious files.

145. Storing data relevant to whether content was suspicious, including results of a comparison involving the operations of the content, was well known and commonplace as of 1996. For example, the user manual for the ThunderBYTE antivirus scanner ("TBAV", EX1006) discloses the generation of such log files. TBAV is a software package used to protect from and detect computer viruses and other malicious software. As discussed above, TBAV includes a utility called "TbLog, which is designed primarily to create log files in response to various TBAV alert messages." EX1006 at 139. TBAV states that the TbLog utility is "primarily for network users," and provides convenient logging of activity so that a "supervisor can easily keep track of what's going on." *Id.*

146. In particular, TbLog is described as a "memory resident TBAV utility that writes a record into a log file whenever one of the resident TBAV utilities

pops up with an alert message. It also records when a virus is detected." *Id*. The log file entries made by TbLog are described as follows: "A TbLog record provides three pieces of information: The time stamp of when the event took place. The name of the machine on which the event occurred. An informative message about what happened and which files were involved. This information is very comprehensive and takes only one line." *Id*.

147. It would have been obvious to store the results of Jaeger's comparison in an event log, such as the one disclosed by TBAV, as when Jaeger's comparison indicates that content is attempting an unauthorized action, this indicates that the content may be malicious. Just as TBAV's TbLog utility records alerts and virus detection events, so would it have been obvious to do the same with Jaeger's comparison results, as a similar function would be served. Moreover, a person of ordinary skill in the art would have had a reasonable expectation of success in doing so, as recording of data in log files has long been a routine activity in computer software.

148. As discussed above, Jaeger in view of CORBA and TBAV discloses each of the elements of claim 5 of the '755 patent. Accordingly, for the reasons discussed above, it is my opinion that Jaeger in view of CORBA and TBAV renders claim 5 obvious.

X. GROUND 3: CLAIMS 6-8 ARE OBVIOUS OVER JAEGER IN VIEW OF CORBA AND ARNOLD

149. As explained in detail below, it is my opinion that each and every element of claims 6-8 of the '755 patent can be found in the prior art, including the references identified below.

150. Claims 6-8 of the '755 patent are presented below in bold text followed by my analysis of each claim. The analysis below identifies exemplary disclosure of the cited references relative to the corresponding claim elements, and it is not meant to be exhaustive.

Claim 6: The system of claim 3, wherein the response engine informs the user when the security policy has been violated.

151. Claim 6 of the '755 patent depends from claim 3. My analysis above in Ground 1 with respect to claim 3 explains how Jaeger in view of CORBA renders obvious each and every element of claim 3.

152. To the extent that Jaeger does not explicitly disclose that response engine informs the user when the security policy has been violated, this would have been an obvious feature to include in Jaeger's system. When Jaeger's system detects a security violation, the corresponding operation is stopped from executing. It would be obvious to alert the user of this for a number of reasons. First, for example, software that attempts unauthorized actions represents a danger, as such an action indicates the possibility that the software is malicious. *See, e.g.*, Jaeger at

0002, 0003, 0015. It would be obvious to inform the user of such danger. Second, when Jaeger's system detects a security violation, it prevents the unauthorized operation from executing. This could lead to unexpected behavior from a program, as operations that were expected to occur would not do so. If the user is not alerted to this fact, the user may not realize that certain expected operations were blocked. Finally, in the case where the software is innocent but has the wrong domain rights, alerting the user could allow the user to correct the rights and allow the program to execute. For any of these reasons, a person of ordinary skill in the art would be motivated to alert the user when the security policy has been violated.

153. Furthermore, alerting the user of potentially malicious content was well known in the art of virus detection. For example, Arnold (U.S. Patent No. 5,440,723, EX1007) discloses a system for detecting "anomalous behavior that may indicate the presence of an undesirable software entity such as a computer virus, worm, or Trojan Horse." EX1007 at Abstract. Arnold teaches that "a number of commercial computer virus scanners are successful in alerting users to the presence of viruses that are known apriori." *Id*. at 64-66. Arnold describes the use of user alert messages when a potential virus or other malicious content is detected, and teaches that on detection of dangerous content, "the use of the distress signal is appropriate to alert the user and/or an expert to the existence of a confusing, potentially dangerous situation." *Id*. at 20:62-64, *see also id*. at Fig. 2 (including

steps to "Alert user" on detection of a virus and "generate distress signal, if required"); EX1006 at 75 (describing displaying a "virus alert window" if a scanner "finds a file to be suspicious"). For example, when a program is blocked due to a security violation, a person of ordinary skill would consider that a "confusing, potentially dangerous situation," in which the user should be alerted.

154. A person of ordinary skill would be motivated to alert the user of security policy violations for any of the reasons discussed above. Furthermore, a person of ordinary skill in the art would have had a reasonable expectation of success in doing so, as generating alert messages was a standard practice in the graphical user interfaces of the mid-1990s and their predecessors.

155. As discussed above, Jaeger in view of CORBA and Arnold discloses each of the elements of claim 6 of the '755 patent. Accordingly, for the reasons discussed above, it is my opinion that Jaeger in view of CORBA and Arnold renders claim 6 obvious.

Claim 7: The system of claim 3, wherein the response engine stores information on the Downloadable in a suspicious Downloadable database when it includes extension calls or operating system calls that are forbidden according to the predetermined security policy.

156. Claim 7 of the '755 patent depends from claim 3. My analysis above in Ground 1 with respect to claim 3 explains how Jaeger in view of CORBA renders obvious each and every element of claim 3.

157. To the extent that Jaeger does not explicitly disclose that the response engine stores information on the Downloadable in a suspicious Downloadable database when it includes extension calls or operating system calls that are forbidden according to the predetermined security policy. However, it would have been obvious to a person of ordinary skill in the art, as storing information in a database relating to potentially malicious content was a known practice in anti-virus technology.

158. For example, Arnold discloses storing information on the Downloadable in a suspicious Downloadable database. Arnold discloses that part of a virus detection process includes "extracting an identifying signature from the executable code portion and adding the signature to a signature database." EX1007 at Abstract. This step is illustrated in Fig. 2 of Arnold, where step E includes extracting a signature from code and adding the signature to a database. The purpose of this storage is to use the stored signatures to detect copies of previously-detected viruses, as well as variants thereof. EX1007 at 9:13-41. The signatures are used in step B of Fig. 2 to detect viruses. *Id.* at 5:28-68. Arnold describes how the signatures recorded to the database are used: "signature report(s) may be displayed on the display device 20 and/ or are written to a database for use by the virus scanner. As soon as the signature for the previously unknown virus has

been extracted, the scanner of Step B uses the signature to search for all instances of the virus in the system." *Id.* at 19:38-43.

159. A person of ordinary skill in the art would be motivated to store information on the Downloadable in a suspicious Downloadable database, as taught by Arnold, because an attempt to perform an unauthorized action indicates that the content may be malicious. *See* Jaeger at 0002. Thus, recording information about the content to a database, such as the database of Arnold, would allow for more efficient future detection of copies of the potentially malicious content (as well as close variants). Moreover, a person of ordinary skill would have a reasonable expectation of success in doing so, as maintaining and updating databases was a common practice in computer science, and Arnold teaches that doing so helps protect a system from malicious or otherwise dangerous content.

160. As discussed above, Jaeger in view of CORBA and Arnold discloses each of the elements of claim 7 of the '755 patent. Accordingly, for the reasons discussed above, it is my opinion that Jaeger in view of CORBA and Arnold renders claim 7 obvious.

Claim 8: The system of claim 3, wherein the response engine discards the Downloadable when it includes extension calls or operating system calls that are forbidden according to the predetermined security policy.

161. Claim 8 of the '755 patent depends from claim 3. My analysis above in Ground 1 with respect to claim 3 explains how Jaeger in view of CORBA renders obvious each and every element of claim 3.

162. To the extent that Jaeger does not explicitly teach that the predetermined responsive action includes discarding the Downloadable, it would have been obvious to do so. When Jaeger's system detects an unauthorized request for an operation, this indicates that the content in question may be malicious. *See, e.g.*, Jaeger at 0002. As malicious content is dangerous, discarding the content, such as by deleting it, is an obvious response when it is detected. Moreover, Jaeger's system includes a file system through which processes "can perform read, write, and execute operations." EX1005 at 0005. Deletion would be an elementary operation in such a file system, performed through an appropriate "write" command. To discard a potentially malicious file in this manner would have been standard practice in 1996.

163. For example, Arnold teaches that malicious content should be discarded when detected. Arnold describes a cleanup process that occurs upon the detection of malicious code: "Cleanup involves killing active worm processes (determined by tracing the process tree), and deleting worm executables and

auxiliary files from storage media." EX1007 at 21:34-37. This corresponds to Arnold's Step B from Fig. 2, which is described as "taking remedial action if [undesirable software entities] are discovered." *Id.* at Abstract. In particular, Arnold describes a "Step B1," in which "the user is alerted, and the virus removed (killed) by traditional methods, such as restoration from backup (either automatically or manually by the user) or disinfection (removal of the virus from all of the software it has infected.)" *Id.* at 5:61-65. Accordingly, Arnold teaches the discarding of malicious code upon detection.

164. A person of ordinary skill would be motivated to discard a Downloadable, as taught by Arnold, when Jaeger's system detected unauthorized behavior, as such unauthorized behavior indicates that the Downloadable is likely to be malicious, and leaving malicious software in place would needlessly expose the system to danger. Moreover, a person of ordinary skill would have a reasonable expectation of success, as Arnold teaches that discarding a potentially malicious Downloadable is one of the "traditional methods" for dealing with malicious software. *Id.*

165. As discussed above, Jaeger in view of CORBA and Arnold discloses each of the elements of claim 8 of the '755 patent. Accordingly, for the reasons discussed above, it is my opinion that Jaeger in view of CORBA and Arnold renders claim 8 obvious.

XI. CONCLUDING STATEMENTS

166. In signing this declaration, I understand that the declaration will be files as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be subject to cross-examination in this case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

167. 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.

Dated: March 1, 2017 By: / Azer Bestavros /
Azer Bestavros

XII. APPENDIX – LIST OF EXHIBITS

Exhibit No.	Description
1001	U.S. Patent No. 9,219,755 to Shlomo Touboul
1003	*Curriculum Vitae* of Dr. Azer Bestavros
1004	File History of U.S. Patent No. 9,219,755 (excerpts)
1005	Jaeger *et al.*, "Building Systems that Flexibly Control Downloadable Executable Content" (July 1996)
1006	ThunderBYTE Anti-Virus Utilities User Manual (1995)
1007	U.S. Pat. No. 5,440,723 to Arnold *et al.*
1008	U.S. Pat. No. 6,167,520 to Shlomo Touboul
1009	U.S. Pat. No. 6,092,194 to Shlomo Touboul
1010	U.S. Provisional Application No. 60/030,639
1011	"The Common Object Request Broker: Architecture and Specification," (December 1993)
1012	U.S. Pat. No. 5,696,822 to Carey Nachenberg

1013	U.S. Pat. No. 6,480,962 to Shlomo Touboul
1014	U.S. Pat. No. 5,867,647 to Haigh *et al.*
1015	U.S. Pat. No. 5,857,102 to McChesney *et al.*
1017	ThunderBYTE Evaluation Request Page (1996)
1018	ThunderBYTE Server Statistics for April, 1996
1019	Proceedings of the Sixth Annual USENIX Security Symposium
1020	USENIX Proceedings Download Page for Jaeger (Nov. 14, 1996)
1021	USENIX Publication Order Form (Nov. 22, 1996)
1022	U.S. Pat. No. 5,754,830 to Butts *et al.*