

Filed on behalf of Symantec Corporation

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL
AND APPEAL BOARD

SYMANTEC CORPORATION

Petitioner

v.

FINJAN, INC

Patent Owner

Case To Be Assigned
U.S. Patent No. 8,677,494

**DECLARATION OF JACK W. DAVIDSON IN SUPPORT OF
PETITIONER PURSUANT TO 37 C.F.R. § 42.120**

**Declaration of Jack W. Davidson
In Support of Petitioner Pursuant to 37 C.F.R. § 42.120**

I, Jack W. Davidson, declare as follows

I. Overview

1. I am over 21 years of age and otherwise competent to make this Declaration. I make this Declaration based upon facts and matters within my own knowledge and on information provided to me by others.

2. I have been retained as an expert witness to provide testimony on behalf of Symantec Corporation (“Symantec” or “Petitioner”) as part of the above-captioned *inter partes* review proceeding (“IPR”), including issues relating to the validity of U.S. patent number 8,677,494 (“the ‘494 patent”), entitled “Malicious mobile code runtime monitoring system and methods.” I also understand that the ‘494 patent was filed on November 7, 2011 and issued on March 18, 2014 and that the ‘494 patent is currently assigned to Finjan, Inc. (“Finjan” or “Patent Owner”).

3. In addition to this Declaration, I have also prepared a separate declaration in support of another IPR petition also involving the validity of the ‘494 patent, which I understand being filed by Symantec concurrently with this Petition and Declaration. As discussed in more detail in my other declaration, it is my understanding that, in the other petition, Symantec is challenging the priority date of the ‘494 patent. For purposes of this Declaration, however, I was asked to

assume that the challenged claims are entitled to the earliest priority date referenced in the '494 patent, *i.e.*, November 8, 1996.

4. I have reviewed and am familiar with the specification and prosecution history of the '494 patent. A copy of the '494 patent is provided as Symantec 1001. I have also reviewed the related patents referenced in the '494 patent specification and certain portions of their prosecution histories, where relevant. As I explain in more detail below, I am familiar with the technology at issue as of the time of the '494 patent, which, for purposes of this Declaration, I have assumed to be November 8, 1996.

5. I have also reviewed and am familiar with the following prior art, which I understand is being used by Symantec in the Petition for *Inter Partes* Review of the '494 patent:

- a. U.S. Patent No. 5,313,616 ("Cline")
- b. *A Sense of Self for Unix Processes*, by Stephanie Forrest *et al.*
("Forrest")
- c. *Dynamic Detection and Classification of Computer Viruses Using General Behaviour Patterns*, by Morton Swimmer *et al.*,
("Swimmer")
- d. U.S. Patent No. 5,623,600 ("Ji")

6. With its corresponding Petition and this supporting Declaration, I understand Symantec is requesting that the Patent Office institute a review of claims 1, 2, 5, 6, 10, 11, 14, and 15 of the '494 patent, and that the requested review is based on the following grounds:

- a. Ground 1: Swimmer anticipates claims 1, 2, 6, 10, 11, and 15 under 35 U.S.C. § 102
- b. Ground 2: Swimmer renders obvious claims 5 and 14 under 35 U.S.C. § 103
- c. Ground 3: Cline in view of Ji renders obvious claims 1, 2, 5, 6, 10, 11, and 15 under 35 U.S.C. § 103
- d. Ground 4: Forrest in view of Ji renders obvious claims 1, 2, 5, 6, 10, 11, and 15 under 35 U.S.C. § 103

7. I have been asked to provide a technical review, analysis, and insight regarding the above-noted references, which I understand form the basis for the grounds of rejection set forth in the Petition.

8. I am being compensated for my time in connection with this IPR at a rate of \$400 per hour. I am also being compensated for any out-of-pocket expenses for my work in this review. My compensation as an expert is in no way dependent upon the results of any investigations I undertake, the substance of any opinion I express, or the ultimate outcome of the review proceedings. I have been advised

that Bryan Cave LLP represents the Petitioner Symantec, Inc. in this matter. I have no direct financial interest in Symantec, Finjan, or the '494 patent.

II. My Background and Qualifications

9. I am a Professor of Computer Science at the University of Virginia. In addition, I am the Founder and President of Zephyr Software LLC. Zephyr Software, in business since 2001, provides a variety of services including innovative computer security solutions targeted mainly for U.S. Department of Defense applications. For more than 35 years, I have been involved in the design of computer systems and software as well as leading and managing large software development projects.

10. I earned a Bachelor's of Applied Science in Computer Science from Southern Methodist University in 1975, a Master's of Science in Computer Science from Southern Methodist University in 1977, and a Doctorate in Computer Science from the University of Arizona in 1981. After receiving my Doctorate, I joined the faculty at the University of Virginia. In addition, I have held visiting positions at Princeton University and Microsoft Research in Redmond, Washington.

11. For over 35 years, I have conducted research in a variety of areas in computer science including compilers, interpreters, programming languages, computer architecture, embedded systems, program analysis, and most recently

computer security. My current research in computer security involves developing methodologies for preventing attacks against critical, enterprise-level computer systems and preventing malware from infecting personal and mobile computers. In these areas and others I have led and managed several large-scale projects involving the collaboration of top U.S. researchers. I am currently leading a large project (\$5.8M) called the Cyber Fault-tolerant Attack Recovery project at the University of Virginia, which has been funded by the Defense Advanced Research Project Agency (DARPA). The goal of the Cyber Fault-tolerant Attack Recovery project is to develop defensive cyber techniques that can be deployed to protect existing and planned software systems without requiring changes to the concept of operations of these systems.

12. I am also the principal investigator of a project funded by the Air Force Research Laboratories (“AFRL”) in Rome, NY. The goal of this project is to transition the results of our previously funded research in cyber security from our research laboratory to the field. That is, we are working with the AFRL to automatically secure mission-critical system against attack by well-funded, determined malicious adversaries and to develop and carry out compelling demonstrations, tests, and exercises that demonstrate the power and effectiveness of the techniques developed in the Dependability Group at the University of Virginia.

13. As my current research focus is in cyber security, I have published extensively in the field of computer security. In addition to other publications, the paper “Safe Virtual Execution Using Software Dynamic Execution” written by Kevin Scott and myself and presented at the 18th Annual Computer Security Applications Conference held in Las Vegas, Nevada in December 2002 is particularly relevant to the matter being considered.

14. My *curriculum vitae*, which is provided as Symantec 1019, lists my publications in the computer security area.

15. In addition to my scholarly activities in the field of cyber security, I am the President and sole owner of Zephyr Software LLC. I founded Zephyr Software as another vehicle for commercializing my research. Currently, Zephyr Software is focused on commercializing cyber security solutions. Including myself, Zephyr Software has four employees. Zephyr Software currently has Phase II SBIR contracts from DARPA and the Office of Naval Research (“ONR”).

16. The DARPA contract is targeted at securing embedded systems. Network routers, communications equipment, supervisory control and data acquisition (“SCADA”) systems, and industrial control systems (“ICS”) are some examples of embedded systems. Because these systems are part of a critical infrastructure, such as plant operations, the power grid, communication systems, transportation systems, and similar operations, it is vital that these systems be

protected from malicious attacks.

17. The work being performed under the ONR contract includes developing techniques to prevent malicious adversaries from taking over the control of a program via a technique known as “program hijacking.” Using program hijacking, a malicious entity can take control of a program to carry out a variety of attacks such as denial of service, secret information leakage, shutdown of critical services, and similar attacks.

18. In addition to my research and commercialization activities, I am also an accomplished and award-winning instructor. In 1989, I received the NCR Faculty Innovation Award for my development of innovative curriculum materials and outstanding teaching. I am the co-author of two widely used introductory programming textbooks, *C++ Program Design: An Introduction to Programming and Object-Oriented Design* and *Java 1.5 Program Design* both published by McGraw-Hill.

19. In 2008, I was co-recipient (with my co-author James P. Cohoon) of the IEEE Computer Society Taylor L. Booth Education Award for “sustained effort to transform introductory computer science education through lab-based multimedia pedagogy coupled with examples that attract a diverse student body.” In addition, I have given invited lectures at the Third International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems held

in L'Aquila Italy in 2007. Approximately 200 students attended this summer school from the member nations of the European Union.

20. As part of my ongoing activities in computer security, I created and teach a course about cyber security at the University of Virginia. The course title is "Defense against the Dark Arts." The course focuses teaching students techniques for defending computers from computer viruses, computer worms, and other types of malicious attacks. The course was first taught in the Fall of 2005 and I have taught it multiple times since that time. I last taught the course in Spring of 2014.

21. I also was a lecturer in the inaugural Indo-US Engineering Faculty Leadership Institute held in Mysore, India. The goal of the Leadership Institute is to improve University education in India. The Institute was attended by 120 faculty members from Indian Universities.

22. In the summers of 2010, 2011, 2012, and 2014, I helped organize and lectured at the International Summer School on Information Security and Protection (ISSISP) held in Beijing, China (2010), Ghent, Belgium (2011), Tucson, Arizona (2012), and Verona, Italy (2014). Each summer school was attended by 50 students from various international universities. ISSISP 2015 will be held in Rio de Janeiro, Brazil.

23. Because of my expertise and stature within in the computing community, I am often asked to serve on important Boards and Councils. I served

as an elected member-at-large of the Association of Computing Machinery (ACM) Special Interest Group on Programming Languages (SIGPLAN) for four years. ACM is the largest professional computing society in the world. I was elected chair of SIGPLAN in 2005. I am a member of the ACM Council, which oversees the operation of ACM, and I am co-chair of ACM's Publications Board, which oversees the publication of the organization's 44 professional journals and 8 magazines, and a professional book series.

24. As a leading expert in the field, I help organize many technical conferences in the area including the International Conference on Parallel Architectures and Compilation Techniques ("PACT"), International Symposium on Code Generation and Optimization ("CGO"), Conference on Programming Language Design and Implementation ("PLDI"), Conference on Languages, Compilers and Tools for Embedded Systems ("LCTES"), International Conference on Compilers, Architectures and Synthesis for Embedded Systems ("CASES"), Conference on the Principles of Programming Languages ("POPL"), International Conference on Autonomic Computing ("ICAC"), and International Conference on High-Performance and Embedded Architectures ("HiPEAC").

25. In the past, I was an Associate Editor of the *ACM Transactions of Programming Languages and Systems* ("TOPLAS") and *ACM Transactions on Architecture and Code Optimization* ("TACO") journals. TOPLAS is the archival

journal in the area of programming languages and compilers. TACO is an archival journal in the area of computer architecture and program optimization. In 2009, I received SIGPLAN's Distinguished Service Award for "substantial and sustained contributions to the programming languages research community and to SIGPLAN in particular."

26. I am a Senior Member of the Institute of Electrical and Electronics Engineers ("IEEE"), the IEEE Computer Society. I am a Fellow of the Association for Computer Machinery ("ACM"). The ACM Council established the ACM Fellows Program in 1993 to recognize and honor outstanding ACM members for their achievements in computer science and information technology and for their significant contributions to the mission of the ACM. The ACM Fellows serve as distinguished colleagues to whom the ACM and its members look to for guidance and leadership as the world of information technology evolves.

27. A more detailed listing of my professional background and accomplishments is found in my *curriculum vitae* provided as Symantec 1019.

III. My Expertise and the Person of Ordinary Skill in the Art

28. As a result of my more than thirty-years' experience in the field of computer science and my deep involvement over the last 15 years with computer security through teaching and research, I am very familiar with techniques to secure and protect computer systems, including techniques to prevent computer

viruses, worms and other types of attacks from corrupting both personal computers and enterprise-level systems.

29. Accordingly, I am qualified to provide expert opinions on the technology described in the '494 patent as well as the teachings of the prior art references at the time of the '494 patent.

30. In my opinion, a person of ordinary skill in the art at the time of the '494 patent would have a Master's degree in computer science, computer engineering, or a similar field, or a Bachelor's degree in computer science, computer engineering, or a similar field, with approximately two years of industry experience relating to computer security. Additional graduate education might substitute for experience, while significant experience in the field of computer programming and malicious code might substitute for formal education.

IV. Applicable Legal Standards

31. I am not an attorney and do not expect to offer any opinions regarding the law. However, I have been informed of certain legal principles relating to patent claim construction and invalidity that I relied upon in reaching opinions set forth in this report.

Obviousness

32. It is my understanding that obviousness is determined from the vantage point of a person of ordinary skill in the art at the time the invention was

made. In order for a claim to be considered invalid under this ground, I understand that the proposed combination of asserted references must teach or suggest each and every claim feature and that the claimed invention as a whole must have been obvious at that time to one of ordinary skill in the art.¹

33. My understanding is that one should avoid the use of “hindsight” in assessing whether a claimed invention would have been obvious. For example, an invention should not be considered in view of what persons of ordinary skill would know today, nor should it be reconstructed after the fact by starting with the claims themselves and/or by reading into the prior art the teachings of the invention at issue.

34. It is my understanding that obviousness cannot be proven by mere conclusory statements or by merely showing that an invention is a combination of elements that were already previously known in the prior art. Rather, it is my understanding that a party challenging a patent in an *Inter Partes* Review

¹ Accordingly, I understand that that the term “obvious” has both a legal and a technical meaning. When the term is used throughout this declaration, my opinions and conclusions will be directed to the technical meaning of obvious (i.e., whether subject matter was within the technical grasp of a person of ordinary skill at the time of the ‘494 patent).

proceeding must further establish by a preponderance of the evidence that there was an apparent reason with some rational underpinnings that would have caused a person of ordinary skill at the time of the invention to have combined and/or altered these known elements to arrive at the claimed invention. Such reasons might include, for example, teachings, suggestions, or motivations to combine that would have been apparent to a person of ordinary skill in the art.

Claim Language

35. I understand that, in *Inter Partes* Review proceedings, claim terms are to be given the broadest reasonable construction in light of the specification as would be read by a person of ordinary skill in the relevant art.

36. As the result of my education and experience, I believe that I understand how the asserted claims of the '494 patent would be understood by a person of ordinary skill in the art applying the above standard.

V. Overview of Relevant Computer Security, Malware Detection, and Internet Technology at the Time of the '494 Patent

37. At the time of the '494 patent, the use of computers was rapidly becoming widespread and commonplace. In particular, companies and other organizations were relying on networked computer systems to perform various tasks, store various information, and manage and control various infrastructure. As the use of such networked computer systems increased significantly, , computer

viruses and other types of malware became a major problem for the computer industry.

38. There were three major factors that contributed to the significant growth in malware. First, sophisticated malware writers had developed tools that allowed relatively unsophisticated programmers to create sophisticated malware. These tools could be easily downloaded using the Internet. A second reason was the growth in computer usage by individuals. Computers had become commodity consumer products. A third reason was the growth of the Internet as computer networks became more ubiquitous. More users, the availability of networking and the development of the World Wide Web (WWW), led to the phenomenal growth of the Internet. In 1993, traffic on the Internet was growing at the incredible rate of 341,000%. As the Internet grew, so did the opportunity for criminals and other malicious entities to use the Internet to spread malware for illicit financial gain.

39. One of the primary ways malicious entities would compromise a computer on the Internet was through a malicious download when a user visited a web page. At the time, typical web-based systems allowed authors to attach an executable program to a Web page, so that anyone visiting the web page automatically downloads and runs the program. Thus, simply visiting a Web page may cause a user to unknowingly download and run a program written by a criminal or other malicious person.

40. Such downloaded executables were written using various programming languages such as Java, ActiveX, VisualBasic, JavaScript, and Web plug-ins. Examples of popular plug-ins include QuickTime (viewing videos), Shockwave (multimedia viewer), and Acrobat Reader (for viewing PDF files).

41. Another commonly used method to deliver an executable to a machine was via an e-mail attachment. Here, an executable was downloaded to the victim's computer via an attachment to an e-mail message. The attachment may be named to appear as if it is a benign text file, image, digital music, etc. In reality, however, the file is an executable that performed the malicious actions intended by the author.

42. Because of the frequency of these types of attacks, there was much interest by industry, university research centers, and government research laboratories in developing techniques to detect and prevent malicious downloads from taking malicious actions such as modifying or destroying files, monitoring the user's online activities, or stealing valuable information.

43. The main defense against various types of malware, including malicious executable programs, was anti-virus software. Such software was generally referred to as anti-virus software even though it would detect other types of malware such as spyware, backdoors, spammers, and keyloggers that might be included or embedded in a malicious, executable program.

44. Initially, the dominant technique used by anti-virus software to detect malware was signature-based scanning. Signature-based scanning is analogous to a common, standard medical approach for determining if a person is infected with a certain biological pathogen. A blood test is performed to see if particular antibodies are present that indicate that the subject is infected. Similarly, with signature-based virus detection, the anti-virus software scans relevant files for a “fingerprint” or “signature” that, if present, indicates malware is present.

45. At the time, anti-virus tools used various approaches to create signatures. One widely used technique was to create a set of patterns to detect viruses and other malware. A pattern might be targeted for a particular family of related viruses. An example of such a pattern is:

SIG: 0x0E,0xBE,Skip(0x02),0x56,0xC3,Skip(0x3),0x83,0xEE,0x1E

46. The signature specifies that the file contains a virus if the file has a sequence of bytes that match the pattern specified. The pattern says look for two consecutive bytes that have the values 0x0E and 0xBE, then skip the next two bytes (their contents are irrelevant), then look for byte values 0x56 and 0xC3, skip three bytes, then look for 0x83, followed by 0xEE and 0x1E.

47. There are many aspects to creating powerful, effective, signature-based anti-virus software. One key aspect for effective scanning is the completeness of the corpus of signatures used by the scanner. If the signature

database does not contain a signature for specific malware, the malware most likely will not be detected. Anti-virus vendors expend considerable effort to ensure their signature databases contain up-to-date signatures of newly discovered viruses and that these updated databases are provided to the licensees of their software on a timely basis.

48. Another aspect of effective scanning is the sophistication of the scanning algorithms and techniques. Anti-virus software vendors continually investigated new scanning techniques to both speed the scanning process and to improve the accuracy. Much like the medical tests I mention above, signature-based scanning may sometimes result in false positives or false negatives. In the medical context, a false positive is when a test has incorrectly indicated the presence of a pathogen when there is, in actuality, none present. A false negative is when the test has incorrectly indicated that no pathogen is present when there is, in actuality, a pathogen present.

49. Another well-known technique employed by anti-virus tools at the time was hashing. Hashing was a well-known technique that was (and still is) used in several contexts of anti-virus technology. One use of hashing is to create a unique “digest” of a file. The message digest is orders of magnitude smaller than the message (hence the term digest). The original use of such hashes was to create a digest of a message to detect if a message was corrupted during transmission to a

receiver. The corruption could be because of an error in the transmission (e.g., a bit or bits are inadvertently changed or dropped) or because the message had been intentionally changed.

50. Before transmitting the message, a cryptographic hash function is used to create the message digest. The digest is attached to the message (either prepended or appended) and the package would be transmitted. At the receiver, after the package was received, the digest would be recomputed and compared to the digest attached to the message. If the computed digest was different from the attached digest, then either the message or the attached digest had been corrupted and appropriate action could be taken. As one example, the receiver could request that the message be resent. Attaching a digest to a message is conceptually similar to attaching a certificate to a downloaded executable.

51. Attaching or appending additional information (such as a certificate) onto executables that were downloaded or transferred via a network was also well known in the art at the time of the '494 patent. For example, Atkinson (U.S. Patent No. 5,892,904, provided as Ex. 1022) teaches that "a publisher digital certificate 122 (FIG. 4) and publisher signature 110 are attached, appended to or incorporated with an executable file 102." Atkinson, col. 6:44-46, FIG. 4. In addition, it was also well known that components of these certificates could be used to link to and retrieve additional information or data related to the executable. Atkinson, col.

2:56-58 (“publisher digital signature also includes an identifying name of the executable file and a link or hyperlink to a description of the executable file”),

FIG. 4.

52. Such cryptographic hashes were also used to create a database on a user’s machine of files that had previously been scanned (perhaps using a signature-based scanner) and were deemed to be virus-free. The database contained a cryptographic hash, and a pointer to the file on disk. Periodically, the anti-virus software would check to make sure the cryptographic hash matched the computed hash. If the computed hash did not match the hash stored in the database, this indicated the file had been changed, which might be the result of a virus. Typically further checking would be done. These databases were often called file integrity databases. Such databases were also used by intrusion detection systems.

53. In the context of anti-virus technology, cryptographic hashes were also used to implement “whitelists” and “blacklists” that could be quickly checked to determine whether to allow an executable to be downloaded and/or run on a computer. To create whitelists, executable files that were known to not contain viruses were hashed, and these hashes were stored in a whitelist table. Similarly, executable files that were known to contain viruses were hashed, and these hashes were stored in a blacklist table. When a new file was received (perhaps it is downloaded from a website, or attached to an e-mail), the hash function was

applied to the file and a hash value was computed. Using the computed hash value, the whitelist and blacklist were searched. If the hash value for the received file was on the whitelist, the file was categorized as not containing a virus. If the hash value for the received file was on the blacklist, the file was categorized as containing a virus.

54. Such hashes were also widely used to create efficient methods to store and find information. Well known to those skilled in the art was the use of hashing techniques to create “hash tables” for efficiently storing and retrieving information. A hash table is a data structure that is searched using the hash value computed by the function (often called a “hash function” in this context). There are several techniques for creating hash tables, but the key idea is that the hash value of an object is used to locate the object in the table. Those skilled in the art routinely used hash tables to store information for fast lookup. Furthermore, it was well understood by those skilled in art that in many cases it was often preferable to store the hash of an object in other data structures rather than storing a duplicate of the object (which required substantially more storage space). The object could easily and quickly be retrieved by using the hash and then accessing the hash table that contained the object.

55. Over the years, anti-virus researchers and researchers in other areas (e.g., software engineering and programming languages) have built various tools to

help them analyze programs including downloads. Such tools can be generally categorized as either static or dynamic analyzers. A static analyzer determines information about a program without running the program. Rather it builds various data structures that can be analyzed to determine various properties of a program. A very common data structure that is useful for static analysis of a program is the control-flow graph.

56. To build a control-flow graph of an executable program, the file containing the program is parsed to determine the instructions that may be invoked by the program. By analyzing the instructions and identifying the instructions that cause the control of flow to change (i.e., jump instructions or control transfer instructions), the static analyzer can construct a graph that represents the relationship between various sections of the program. The nodes in the control-flow graph were often referred to as basic blocks by those skilled in the art.

57. Using the control-flow graph, a static analyzer performs other useful analyses. One example is called “dead code” elimination. Here, code that cannot possibly be executed can be removed from the program thereby saving space.

58. While static analysis is a powerful tool, it must be conservative because it is analyzing the program without the benefit knowing what inputs the program might process, and therefore could produce erroneous information. Consequently, dynamic analysis was also used to analyze executable programs.

Dynamic analysis requires running the program on some input. The advantage of dynamic analysis is that one can observe how the program behaves on a particular input or set of inputs and monitor the instructions that are called by the program.

59. Because of their complementary nature, both static and dynamic analyses were often used together when analyzing executable programs.

60. At the time of the '494 patent (and even now), anti-virus researchers continually worked to improve the accuracy of the signature-based scanning by lowering the rates of false positives and false negatives. Unfortunately, virus writers also continually worked to create new techniques for creating malware that would evade detection by signature-based scanning. This back-and-forth struggle between virus writers and anti-virus defenders is much like an arms race. Each time a virus writer devised a new mechanism for avoiding detection, anti-virus researchers responded by developing new techniques. With each successive generation of malware, maintaining the effectiveness of signature-based scanning (i.e., low rates of false positives and false negatives) grew more difficult.

61. One of the advantages of signature-based malware detection is that anti-virus vendors could extensively test their signatures to avoid false positives. They did this by maintaining an extensive corpus of benign programs that would be typically found on target machines. Before a new set of signatures is released, the signatures are extensively tested against the corpus.

62. A disadvantage of signature-based scanning is that it is only effective against known families of malware. That is, appropriate signatures can be tested and evaluated only if a sample of the malware has been captured and analyzed. Because of the growth of the Internet, it became easier to create new malware and spread it. Therefore, anti-virus researchers began to search for other approaches that could complement signature-based scanning.

63. Another approach to detecting malware is to monitor a program to detect intrinsic malware behavior, which was commonly referred to as behavior blocking. This approach is based on the hypothesis that malware activity involves abnormal use of the system; therefore security violations could be detected from abnormal patterns of system usage. The approach had been known since at least 1987 where it had been proposed and used for intrusion detection and detection of violations of security policies.

64. One form of behavior blocking was to monitor a program as it runs and observe its actions. If the program behaved in some suspicious way, perhaps by carrying out a set of operations that is characteristic of malware, a predetermined security policy could be applied.

65. To avoid damage that might be done before the malicious behavior is recognized, the monitored program was often executed in a “sandbox.” A sandbox prevents the application from compromising the host computer. The sandbox could

be created through some form of virtualization that is either process-level or system-level on the host machine or a separate machine.

66. One of the first works to propose this concept was that of Wahbe *et al.*, “Efficient Software-based Fault Isolation,” (provided as Symantec 1023). The paper was provided at the ACM SIGOPS (Special Interest Group on Operating Systems) Symposium on Operating Systems Principles in December 1993. The term “sandbox” is used in the paper, but it was the subsequent release of Java in 1995 and its use of the term sandbox that popularized the term.

67. Typically, the monitoring necessary for behavior blocking was done by some type of reference monitor. A reference monitor works by monitoring a program’s execution steps. The execution may be monitored at different levels. It may be low-level and fine-grained, monitoring every instruction or every memory-reference. It may also be high-level, monitoring only certain key system calls, API calls, or application function or method calls.

68. A common approach to monitoring a program’s execution was to rewrite function calls so that a substitute or monitoring function is called instead of the original function. The substitute function can perform the necessary actions to enforce an execution policy (including a security policy). Such actions include recording context information (e.g., the contents of the run-time stack, the values of arguments, etc.), checking, modifying, or recording the values of arguments to

functions, preventing the execution of the original function, and redirecting control to a another substitute function. Because the substitute function can be written in a high-level language, the writer of the substitute function has great flexibility in choosing what actions to take when the substitute function is invoked.

69. Many of these concepts are concretely illustrated in the paper “Safe Virtual Execution Using Software Dynamic Translation” which I coauthored with Kevin Scott. Published in 2002 (provided as Ex. 1024), the paper discusses a novel approach to execution monitoring or behavior blocking based on software dynamic translation (SDT). SDT uses the underlying hardware to execute instructions as opposed to an interpreter or emulator. Thus, it can be made very efficient. This efficiency gives system architects more flexibility in choosing the level of monitoring to be performed. The paper describes how an application’s execution can be monitored so that system calls are intercepted or “hooked” and a specific security policy is enforced.

70. At the time of the ‘494 patent, a common way to enforce computer security was through specifying Access Control Lists (ACLs). These ACLs regulate access to security-sensitive aspects of the computer (e.g., files or network access). A program may be distributed along with an Access Control List specifying a sort of “wish list” for the program. This “wish list” specifies (or is an attempt to specify) which security-sensitive aspects of the computer the program

will attempt to access. For example, if the program will attempt to write the registry, that function would be captured in the program's ACL. When used in this manner, an ACL is a form of "security profile" for a particular program because it is associated with the program and identifies what operations that program may attempt to perform.

71. ACLs may also be used to enforce limits on security sensitive aspects of the computer. For example, an administrator of a network or computer may not allow third party programs to write to the registry. An ACL can be written to restrict such access. For example, if the program described above attempted to access the registry, it would be blocked by such an ACL. This can be enforced regardless of whether, the program was distributed with an ACL or not. When used in this manner, an ACL is a form of "security policy" because it used by the computer system to limit access by programs to certain computer resources. Such ACLs were often stored in a database that was managed by a security administrator so that the policies could be centrally managed.

72. In other words, depending on their application, ACLs could be used to create security profiles (e.g., a profile identifying the operations a program may attempt to perform) or to enforce security policies (e.g., polices restricting the actions of a program, user, or network). In fact, the '194 patent, which is referred to in the '494 patent, recognizes this dual nature of ACLs. U.S. Patent No.

6,092,194 (“the ‘194 patent,” provided as Ex. 1013), col. 6:13-24 (“ACL comparator.... compares the DSP data of the received Downloadable against the access control lists in the received security policy”), col. 8:24-28, claim 55.

73. Another concept closely related to behavior blocking was intrusion detection. The idea behind intrusion detection is that exploitation of the system involves abnormal use of the system. For example, a computer virus might cause an increase in the frequency of executable files being written or modified, or it may effect an increase in the number of changes to the system registry. Similarly, a computer worm might generate excessive network traffic. Such behavior could indicate that the system has been compromised.

74. One of the key aspects of intrusion detection is to determine what constitutes normal behavior of the system. At the time of the ‘494 patent, one approach was to monitor a system that is known to not be compromised and record the behavior of the system. This record of behavior typically consisted of logs of various system behaviors (these logs were also sometimes referred to audit records). Various types of activities can be recorded such as accesses to the system registry or changes/accesses to other key files (e.g., the file containing user/system passwords), etc. Using these audit records, a model of normal behavior was generated. There were numerous methods for creating the behavior model but one

of the most common was statistical techniques to determine the likelihood an event occurring over a period of time.

75. To monitor a system for possible compromise, the behavior of the system is again monitored and events are recorded. As events are recorded the intrusion detection system determines if the current behavior falls within the parameters of what has been determined to be normal behavior. If the behavior of the system deviates from normal behavior (i.e., the observed behavior is outside the statistical profile of normal behavior), the system is determined to be compromised. As one skilled in the art would have readily understood, the system being monitored could be a group of machines, a single machine (e.g., a desktop PC), or a particular program. Furthermore, one skilled in the art would have considered that a statistical profile of normal behavior constitutes a form of security profile because deviations from this profile indicate anomalous behavior (e.g., a virus).

VI. OVERVIEW OF THE '494 PATENT

A. The Specification

76. The '494 patent is concerned with the protection of computers from potentially undesirable or suspicious software programs or code received over a network, referred to as "Downloadables." '494 patent, Abstract, col. 1:59-63, 2:22-3:9. A Downloadable is any "received information [that] includes executable

code.” ‘494 patent, col. 3:3-8, col. 4:5-14, col. 5:64-6:2, col. 9:46-52, col. 15:22-39. Examples of Downloadables in the patent include distributed components, Java applets, JavaScript scripts , ActiveX controls, VisualBasic scripts, zip files and add-ins. ‘494 patent, Abstract, col. 2:22-30 & 59-64, col. 9:46-52.

77. Beyond this high-level and limited discussion of “Downloadables,” the specification of the ‘494 patent does not include much further description of the subject matter in the challenged claims. This subject matter, however, appears to be described in certain other applications referred to in the ‘494 patent. *See* ‘194 patent; U.S. Provisional Patent Application No. 60/ 205,639 (“the ‘639 provisional,” provided as Ex. 1002).

78. These other applications explain that a Downloadable is “received from [an] external computer network” and delivered to a “code scanner.” ‘194 patent, col. 4:33-40, 5:36-42. For “unknown” Downloadables, the code scanner generates Downloadable Security Profile (DSP) data for the Downloadable by “us[ing] conventional parsing techniques to decompose the code (including all prefetched components) of the Downloadable into the DSP data.” ‘194 patent, col. 5:41-45, col. 9:20-42, FIG. 7. The DSP data “includes the fundamental computer operations included in each known Downloadable 307, and may include, READs, WRITEs, file management operations, system management operations, memory management operations and CPU allocation operations.” ‘639 provisional, p. 18, l.

9-13, p. 24, l. 19-p. 25, l. 2 (describing loop commands such as “goto”, “while” “if”, “than” or the like as further examples of potentially suspicious commands); ‘194 patent at col. 5:45-6:3, col. 9:20-42.

79. The Downloadable and its DSP data may then be stored (e.g., in a security database). ‘639 provisional, p. 20, l. 12-16 (“the non-hostile Downloadable is stored in known Downloadable’s 307 and its corresponding DSP data is stored in DSP data 310.”), p. 22, l. 15-21, p. 17, l. 13-19 (describing items 307 and 310 as portions of a “security database”); ‘194 patent, col. 6:9-12.

80. Based on this disclosure, one of ordinary skill in the art would have understood that the DSP data represents an assessment of the Downloadable that identifies the fundamental computer operations (e.g., potentially suspicious system operations) that the Downloadable may attempt to call.

81. This DSP data can then be compared against “security policies” (e.g., at a network security system), before allowing the Downloadable to execute. ‘639 provisional, p. 20, l. 2-12; ‘194 patent, col. 6:13-24 (pass or fail Downloadable using an ACL comparator to compare DSP to security policy). Generally, a “security policy” is a set of rules that are specified by an organization or user. The “security policy” can be used to determine whether a Downloadable (and/or the operations being invoked thereby) should be blocked or allowed to execute based upon its “security profile” (i.e., the DSP data). This allows the same “security

profile” (and its associated Downloadable) to be treated differently depending on the particular “security policy” being enforced. That is different organizations can have different policies, which may treat a Downloadable and its DSP differently.

82. For example, if a Downloadable may attempt a registry write, the registry write would form a part of the DSP, and a client executing the Downloadable would then check the DSP against its own security policies. If the policy does not allow registry writes, the Downloadable may not be allowed to execute. If such registry writes are allowed under the policy, the Downloadable may execute (depending on the rest of the policy and DSP).

83. In my opinion, it is clear that the ‘494 patent does not disclose anything new with respect to generating security profile data (i.e., a list of potentially suspicious operations) for an executable. Instead the patent relies on well-known techniques for deriving such data, such as by parsing and decomposing executable code. ‘639 provisional, p. 19, l. 16-20; ‘194 patent, col. 5:42-45. Thus, the only alleged differences in the challenged claims appear to be that this security profile data is generated for “an incoming Downloadable” and is then stored in a database. ‘494 patent, claim 1. In my opinion one of ordinary skill in the art would have found these concepts, i.e., receiving or intercepting Downloadables over a network and storing data in a database to be simple and

straightforward. These features were well-known long before the time of the '494 patent (e.g., in the references discussed below). *See also supra* at ¶¶ 38-42, 51, 72.

VII. Construction of Certain Claim Terms

A. “Database”

84. It is my understanding that, in the Petition, Symantec has proposed that the broadest reasonable interpretation of the claim term “database” is: “an organized collection of data.” I agree with this construction. In my opinion, this is consistent with the plain and ordinary meaning of the term “database,” as it would have been understood by one of ordinary skill in the art at the time of the '494 patent.

85. For example, Webster’s New World Dictionary of Computer Terms describes a database as a “coherent collection of data.” Ex. 1016, p. 95. Similarly, Webster’s Ninth New Collegiate Database describes a “database” as a “collection of data organized esp. for rapid search and retrieval (as by a computer).” Ex. 1015, p. 325. As another example, Webster’s College Dictionary describes a database as a “collection of organized, related data.” Ex. 1014, p. 339. These contemporaneous definitions are consistent with the construction above.

86. Also, the '494 patent does not attempt to define, limit, the operation of, or provide any particular structure for the claimed “database.” Instead, the patent and the claims themselves only describe the types of data stored within the

database, such as a Downloadable Security Profile (DSP). ‘194 patent, col. 3:47-50 (“[t]he data storage device 230 stores a security database 240, which includes security information . . .”); col. 4:14-18; col. 9:52-55, FIGS. 2, 3; ‘494 patent, claim 1 (“storing the Downloadable security profile data in a database”). In my opinion, in view of this disclosure, one of ordinary skill in the art would have considered the “database” to be “an organized collection of data.”

VIII. Analysis of the Prior Art

A. Swimmer

87. Swimmer describes a computer system, called Virus Intrusion Detection Expert System (VIDES), for detecting and classifying computer viruses. Swimmer, Title. For example, Swimmer explains that this VIDES system can be used “as a type of firewall for programs entering a protected network,” i.e., programs downloaded over a network. Swimmer, p. 13. In order to detect viruses or virus behaviors, Swimmer discloses using an emulator to monitor the activity of a virtual PC, including application programs and code being executed by the PC. Swimmer, p. 1. The emulator creates a stream of system activity data, which includes operations and functions that these programs attempt to invoke. Swimmer, p. 1, 7. Swimmer explains that the activity data is recorded and organized in a database. Swimmer, p. 9 (“<code segment, RecType. StartTime, EndTime, function number, arg (.. },ret(..)>”). This structured data is then used

by an expert system (e.g., ASAX) to detect viruses by employing rules that model typical virus behavior. Swimmer, p. 2, 4-5, 10-12.

1. Swimmer discloses a computer-based method and a system for managing Downloadables

88. Swimmer describes a computer system called “VIDES”, which is “comprise[d] of a PC emulation and an IDES-like expert system.” Swimmer p. 2, FIG. 4. Swimmer also teaches methods for detecting viruses using this VIDES system. Swimmer, p. 1 (“The resulting system is called VIDES: it is a prototype for an automatic analysis system for computer viruses.”). Swimmer explains that in this VIDES system an emulator monitors and records the operations of a virtual computer and an expert system then analyzes the recorded data using rules associated with virus behavior. Swimmer, p. 1 (“an emulator is used to monitor the system activity of a virtual PC [and] the expert system ASAX is used to analyse the stream of data whicg (sic) the emulator produces [using] general rules to detect real viruses generically and reliably, and specific rules to extract details of their behaviour.”), p. 4-7, 10, 12 (describing exemplary rules), p. 8-10 (describing the use of the emulator to develop/record system activity information), p. 11-12 (describing the application of the expert system ASAX to rules and recorded data).

89. The VIDES system is used to detect viruses in application programs and program code by monitoring and analyzing the functions and operations these

programs attempt to invoke. Swimmer, Abstract, p. 7 (“prerequisite for using an Intrusion Detection (ID) system like ASAX is an audit system which securely collects system activity data”). These application programs can include “programs entering a protected network” (i.e., executable code being downloaded over a network). Swimmer, p. 13.

90. Thus, Swimmer discloses a computer-based method and a system for managing Downloadables (e.g., application programs and executable code received over a network). *See also* ‘494 patent, col. 2:59-3:8 (stating that “application programs” and “executable code” are examples of “Downloadables”), 9:46-52 (same).

2. Swimmer discloses [a receiver for] receiving an incoming Downloadable

91. As discussed above, Swimmer describes methods for detecting viruses in application programs and program code using its VIDES system. Swimmer, p. 1 (“The resulting system is called VIDES: it is a prototype for an automatic analysis system for computer viruses.”). This VIDES system includes an emulator, which monitors the programs and executable code and records certain operations and functions that they attempt to invoke. Swimmer, Abstract, p. 8 (“The solution which was finally chosen was the software emulation of the 8086 processor”). Thus, Swimmer discloses techniques for monitoring and analyzing application

programs and executable code (i.e., Downloadables). *See also* '494 patent, col. 2:59-3:8, 9:46-52.

92. Swimmer also discloses that these “incoming Downloadables” can be received over a network. As an example, Swimmer explains that the VIDES system can be used in a networked environment as part of a firewall for a protected network (e.g., an intranet). Swimmer, p. 13 (explaining that VIDES could be used “to detect viruses in a real environment” and that “[o]ne possibility is to use it as a type of firewall for programs entering a protected network.”). In other words, Swimmer discloses that VIDES can be used at a firewall in order to monitor and analyze incoming Downloadables received at the firewall (e.g., programs that are being downloaded by or sent to a computer located on the protected network).

93. As was well-known in the art a firewall is a security device that was typically be located at a strategic point in a network, such as the connection point between a wide area network (e.g., an Internet) and an internal network of client computers (e.g., intranet).

94. Since the VIDES system can be used at a firewall, one of ordinary skill would have understood this system included a “receiver” for receiving the Downloadable, i.e., standard components for receiving data over the networks such as a network interface card (NIC) or modem, and network protocol software (e.g., a TCP/IP stack). Indeed, one of ordinary skill in the art would have understood

that some form of “receiver” would have been necessary in order to receive the programs over a network.

95. Additionally, even if it is argued that Swimmer does not expressly disclose a receiver for receiving an incoming Downloadable, one of ordinary skill in the art would have clearly found this to be a natural and obvious application of the system described in Swimmer. It would have been obvious that Swimmer’s VIDES system could be used at a network device, such as a gateway or FTP or Web server in order to intercept incoming Downloadables and analyze them before they are sent to a destination computer (e.g., a client computer). Indeed, Swimmer discloses that its VIDES system can be implemented at a firewall device to analyze programs before they enter a protected network. As I explain above, techniques for receiving or intercepting Downloadables were well-known before the time of the ‘494 patent. *See, e.g.*, Ji, col. 5:28-38 (describing a system for performing virus detection on executable files, which can be implemented “on a FTP server or a world wide web server for scanning files and messages as they are downloaded from the web.”); *see also* ‘494 patent, col. 2:22-44 and ‘194 patent, col. 1:24-57 (admitting the concept of “receiving a Downloadable” was well-known in the art).

96. One of ordinary skill in the art would have been motivated to use Swimmer’s system in this manner for various reasons, such as to more efficiently check for viruses by verifying the behavior of incoming “Downloadables,” at a

network connection point for one or more computers on a particular network (e.g., an Intranet). Ji, col. 2:12-29 (teaching that scanning software on individual machines rather than a gateway can be inefficient). For one of ordinary skill in the art, this would have merely amounted to combining well-known prior art elements (*i.e.*, a gateway with Swimmer's VIDEs system) according to well-known software programming techniques in order to yield a predictable result (*i.e.*, a gateway scanner that receives Downloadables and analyzes their behavior).

97. Also, as I explain above, when using Swimmer's system in this manner, one of ordinary skill in the art would have understood that the system would include components (e.g., network cards and modems) for receiving the Downloadables over the networks (*i.e.*, a receiver). Such networking components were conventional and well-known long before the time of the '494 patent.

3. Swimmer discloses [a Downloadable scanner for] deriving security profile data for the Downloadable, including a list of suspicious computer operations that may be attempted by the Downloadable

98. As discussed above, Swimmer teaches that the VIDES system uses an emulator to monitor application programs and code and generate a stream of system activity data. Swimmer, p. 7 ("The prerequisite for using an Intrusion Detection (ID) system like ASAX is an audit system which securely collects system activity data."). The emulator, in order to produce this data stream "accepts the entire instruction set of a processor as input, and interprets the binary code as

the original processor would.” Swimmer, p. 8, p. 9 (“audit record attributes of records as collected by the PC emulator have the following meaning... [t]he final format for an MS-DOS audit record is as follows: <code segment, RecType. StartTime, EndTime, function number, arg (.. },ret(..)>”).

99. Swimmer explains that the “audit system was integrated into an existing PC emulation by placing hooks into the module for processing all opcodes corresponding with the events.” Swimmer, p. 9. Swimmer’s audit system and/or emulator generates audit records (i.e., Downloadable security profile data) for the Downloadables that identifies and lists functions (i.e., operations) attempted by the Downloadable. An example of these audit records is illustrated in figure 3 of Swimmer.

100. Swimmer provides further details regarding these audit records. These audit records include a field, called “function number,” which is the “number of the DOS function requested by the program.” Swimmer, p. 9. One of ordinary skill in the art would have known that, in DOS, function numbers are assigned to “INT 21h” functions. Swimmer, p. 7. These “INT 21h” include various types of system operations. Advanced MS-DOS (relevant excerpts provided as Ex. 1020), p. 7 (“Most of the MS-DOS operating system services are invoked through software interrupt 21h”).

101. For example, function numbers 0, 49, and 76 relate to operations that terminate a program. *Advanced MS-DOS*, p. 9. In addition, a number of these operations relate to the filesystem. For example, function number 15 opens a file and the function number 19 deletes a file. *Advanced MS-DOS*, p. 9-10. Function numbers 72, 73, 74, and 88 are memory allocation operations. *Advanced MS-DOS*, p. 11. Function numbers 68, 94 and 95 are assigned to network operations. *Advanced MS-DOS*, p. 11.

102. These operations identified by the audit system are the same types of operations described as “suspicious operations” in the ‘494 patent. ‘639 provisional, p. 18, l. 9-13 (DSP data “includes the fundamental computer operations,” in a Downloadable such as “file management operations, system management operations, memory management operations and CPU allocation operations.”).

103. Swimmer also discloses that a Downloadable scanner (e.g., an emulator and/or audit system) derives this Downloadable security profile data. Swimmer, p. 8 (the emulator is “a program which accepts the entire instruction set of a processor as input, and interprets the binary code as the original processor would.”). Similarly, the ‘194 patent explains that “code scanner 325 in step 710 resolves a respective command in the machine code.” ‘194 patent, col. 9:24-25.

104. One of ordinary skill would have understood this disclosure to mean that each instruction was decomposed by Swimmer's emulator and/or audit system (e.g., in order to accurately emulate a processor). In turn, this means that the instructions must be properly located in memory according to their alignment. In addition, the opcode for the instruction must be determined along with its parameters, such as registers (and their values). An emulator then takes additional steps and performs the instruction faithfully (or as near faithfully as feasible and possible).

105. Thus, one of ordinary skill in the art would have considered the emulation described in Swimmer to be disclosing a form of dynamic analysis, which enables the host computer to be somewhat secure from the emulated system that is running the virus by virtue of the attendant protections offered by emulation.

106. The emulator (i.e., Downloadable scanner) would also be coupled to the receiver (e.g., the network components at the firewall for receiving the Downloadable). For example, both components would be located on the same computer system (e.g., a firewall system) and would be stored together in memory, such as RAM. '194 patent, col. 3:23-46 (describing the same form of "coupling" for the "code scanner" and "receiver"), FIG. 3. In addition, one of ordinary skill in the art would have understood that, because Swimmer discloses the data representing the Downloadable executable code flows from the receiver on the

firewall to the emulator for analysis, these components must be coupled in some manner.

4. Swimmer discloses [a database manager for] storing the Downloadable security profile data in a database

107. As I discuss above, Swimmer discloses an emulator and/or audit system that monitors Downloadables and generates audit records that list suspicious operations (i.e., security profile data). Swimmer also explains that these audit records are stored in a database. Swimmer, p. 9 (“The final format for an MS-DOS audit record is as follows: <code segment, RecType, StartTime, EndTime, function number, arg (.. },ret(..)>”). For example, Figure 3 (partially reproduced below) illustrates one of these audit records.

```
<CS=3911 Type=0 Fn=30 arg() ret( AX=5)>  
<CS=3911 Type=0 Fn=29 arg() ret( BX=128 ES=3911)>  
<CS=3911 Type=0 Fn=64 arg( AL=61 CL=3 str1=*.COM) ret( AL=0 CF=0)>  
<CS=3911 Type=0 Fn=51 arg( AL=0 str1=COMMAND.COM) ret( AL=0 CX=32 CF=0)>  
<CS=3911 Type=0 Fn=51 arg( AL=1 str1=COMMAND.COM) ret( AL=0 CX=32 CF=0)>  
<CS=3911 Type=0 Fn=45 arg( AL=2 CL=32 str1=COMMAND.COM) ret( AL=0 AX=5 CF=0)>  
<CS=3911 Type=0 Fn=73 arg( BX=5) ret( CX=10241 DX=6206 CF=0)>  
<CS=3911 Type=0 Fn=27 arg() ret( CX=5121 DX=8032)>
```

108. As shown in Figure 3, the audit record includes a list of suspicious operations identified by the audit system that are organized according to a clearly defined structure with various fields (i.e., an organized collection of data). Thus, in my opinion, one of ordinary skill in the art would have considered Swimmer’s audit records to be a form of database (e.g., a flat-file database).

109. Swimmer also discloses that the audit records stored in the database are used by other processes, meaning they “serve one or more applications.” For example, the database is used by an expert system, which is an application that analyzes program behavior using virus behavior rules. Swimmer, p. 1 (“the expert system ASAX is used to analyse the stream of data whicg [sic] the emulator produces”), p. 2 (“ASAX . . . analyse[s] arbitrary sequential files . . . , [which] is the activity data record produced by the PC emulator”).

110. In Swimmer, the audit system (or a portion of it) stores the security profile data (e.g., audit records) in the database. Thus, one of ordinary skill in the art would have understood this to mean that the audit system (or a portion thereof) was a “database manager” (i.e., a component that manages and controls the storage and retrieval of data in the database). Also, this database manager is coupled to the Downloadable scanner (e.g., emulator and/or audit system). For example, both components are located on the same computer system (e.g., a firewall system) and would be stored together in memory, such as RAM. ‘194 patent, col. 3:23-46 (describing the same form of “coupling” for the “code scanner” and “receiver”), FIG. 3. In addition, because Swimmer discloses that the data representing the Downloadable executable code flows from the receiver on the firewall to the emulator for analysis and then to the audit system for storage in the database, one

of ordinary skill in the art would have understood that these components must be coupled in some manner.

111. Additionally, even if it is argued that Swimmer's audit records do not expressly disclose the claimed "database," it would have been obvious that this security profile data could have been stored in any suitable format or structure. For example, one of ordinary skill would have readily understood that this data could easily be stored in any number of well-known formats, such a plain-file, flat-file database, relational database, raw disk, excel spreadsheet, etc. Thus, in my opinion, choosing a particular logical format/organization for how this security profile data is stored in Swimmer (e.g., a relational database storing security profile data for multiple Downloadables) would have been a simple design choice well within the knowledge of ordinary skill in the art at the time of the '494 patent. One of ordinary skill in the art would have been motivated to use such a database for various reasons, such as to improve the organization, efficiency and speed associated with storing, maintaining, and retrieving this data.

112. One of ordinary skill in the art would have also found it obvious to use a database manager with these types of databases. Database managers were well-known at the time of the '494 patent and were routinely used with many different types of databases to manage and/or control the storage and retrieval of data. *See, e.g.,* Cline, col. 10:39-57. These database managers were used to

prevent or control access to the database and to prevent corruption of the database. Data corruption can occur if there are two or more concurrent processes seeking to write to the database. A database manager would mediate access in order to prevent corruption.

113. Thus, one of ordinary skill in the art would have been motivated to use a database manager in Swimmer in order to control access to and/or prevent corruption of the security profile data. For one of ordinary skill in the art, this would have simply involved combining well-known elements (i.e., databases and database managers) using conventional software programming techniques to achieve a predictable result (e.g., a database storing security profile data that was managed by the database manager).

114. Accordingly, for at least the reasons above, it is my opinion that Swimmer discloses all of the limitations in claims 1 and 10 of the '494 patent.

5. Swimmer discloses [the database manager] storing a date & time when the Downloadable security profile data was derived in the database

115. As I discuss above, Swimmer discloses storing audit records that include a list of suspicious operations (i.e., Downloadable security profile data) in a database. Swimmer also explains the entries in the audit records include fields called "StartTime" and "EndTime." Swimmer, p. 9, FIG. 3. Swimmer clarifies that certain fields were omitted for readability. Swimmer, p. 10. These

“StartTime” and “EndTime” entries indicate when the audit record was generated by the emulator and/or audit system (i.e., when the Downloadable security profile data was derived by the Downloadable scanner).

116. Accordingly, for at least the reasons above, it is my opinion that Swimmer discloses all of the limitations in claims 2 and 11 of the ‘494 patent.

6. Swimmer discloses that the suspicious computer operations include calls made to an operating system, a file system, a network system, and to memory

117. As I discuss above, Swimmer discloses that the emulator and/or audit system identifies and records DOS system calls (i.e., suspicious operations) that a Downloadable attempts. Swimmer, Figure 3. As explained above, different functions numbers are assigned to the different types of system calls (e.g., INT 21h functions). Advanced MS-DOS, p. 7 (“Most of the MS-DOS operating system services are invoked through software interrupt 21h”).

118. For example, a number of functions relate to the filesystem. For example, function number 15 is an open file operation and function number 19 is a delete file operation. Advanced MS-DOS, p. 9-10. Function numbers 72, 73, 74, and 88 are memory allocation operations. Advanced MS-DOS, p. 11. Function numbers 68, 94 and 95 are assigned to network operations. Advanced MS-DOS, p. 11.

119. In my opinion, one of ordinary skill would have considered all of these system calls described in Swimmer to be “operating system operations” because they are handled by the DOS operating system. Swimmer also discloses listing specific types of “operating system” calls (e.g., kill thread/process calls). ‘194 patent, col. 5:66-6:3. For example, function numbers 0, 49, and 76 relate to specific operations used by the operating system to terminate a program. Advanced MS-DOS, p. 9. In addition, function number 49, Terminate and Stay Resident (TSR), is typically used by viruses.

120. Accordingly, for at least the reasons above, it is my opinion that Swimmer discloses all of the limitations in claims 6 and 15 of the ‘494 patent.

7. Swimmer teaches that the Downloadable includes program script

121. As I discuss above, Swimmer discloses receiving and deriving security profile data for a Downloadable. Swimmer explains that this VIDES system can be used to derive security profile data for application programs and code, including programs received at a firewall. Swimmer, Abstract, p. 13 (“use it as a type of firewall for programs entering a protected network”).

122. As such, one of ordinary skill would have found it obvious to use these techniques on program scripts. A program script is merely one particular form of executable code that was commonly used at the time of the ‘494 patent. *See e.g.*, Apperson (U.S. Patent No. 5,978,484, provided as Ex. 1021), col. 4:9-10

(recognizing a script is one form of executable code: “code might be in the form of textual scripts, byte codes, P-code, or binary object code”). Client-side scripts such as Javascript, VBscript, and Python were well-known by the time of the ‘494 patent. The ‘494 patent admits that various kinds of program scripts, including scripts received over a network, were well-known and disclosed in a number of prior art references. ‘494 patent, col. 2:22-27.

123. As explained above, Swimmer teaches that VIDES could be “used as a firewall for programs entering a protected network.” Swimmer, p. 13. It was well known in the art that program scripts were often included in files and messages transmitted through firewalls. For example, scripts may traverse firewalls when transmitted via the web to a client. Client-side scripts such as Javascript, VBscript, and Python were well-known by the time of the ‘494 patent.

124. Thus, in my opinion, it would have been obvious to one of ordinary skill in the art that the systems and methods used to receive and derive security profile data for application programs and code taught in Swimmer could also be used for other types of Downloadables, such as program scripts. For such a person, this would have merely involved applying the same techniques disclosed in Swimmer to another well-known form of executable code (e.g., receiving program scripts at a firewall and using an emulator and audit system to identify and record suspicious operations in the script). One of ordinary skill would have been

motivated to do so for various reasons, including to improve the effectiveness of the virus detection systems taught by Swimmer by enabling it to be used with a wider range of Downloadables.

125. Accordingly, for at least the reasons above, it is my opinion that Swimmer teaches the additional limitations in claims 5 and 14 of the '494 patent.

B. Cline and Ji

126. Cline describes software designed to test application programs. Cline, col. 1:9-11, Abstract. Cline is primarily concerned with ensuring portability of a particular piece of application software by “verify[ing] that application software [was] developed in conformance with a set of accepted design rules.” Cline, col. 1:-50-55, *see generally* cols. 1-2, col. 3:38-60. Cline’s verification uses “a static analysis and a dynamic analysis of the application program.” Cline, col. 3:6-7. To perform these different analyses, a conformance database is created which “includes allowable external calls, such as system calls and procedure calls.” Cline, col. 3:7-10, col. 6:34-50, 7:55-65, FIGS. 4-5. Examples of calls listed in the conformance database include: getpid(), open(), window calls, and/or networking calls. Cline, col. 7:44-53, col. 8:27-39, Table 3.

127. Cline explains that the static analysis (SBV) “analyzes the object code of an application program [for] any illegal or erroneous external calls.” Cline, col. 3:10-13, col. 9:43-66. This “static analysis converts an object code version of the

application program into a graph [which is analyzed] for errors and external calls which do not conform to the system rules” that are specified in the conformance database. Cline, col. 3:22-28, col. 11:22-30. Then a graph creating process identifies “basic blocks” of code, which “is a collection of code which has one entrance, one or two exits and in which all instructions between its entrance and exit are executed.” Cline, col. 9:67-10:5. As part of verification, it is determined if code in a basic block contains external calls (e.g., system calls or calls to a library routine). If those calls are present they are then analyzed for conformance with the conformance database. Cline, col. 12:50-59, col. 13:44-53.

128. After static analysis, a dynamic analysis (DBV) “analyzes the application program as it is being run to determine any runtime errors in the calls made by the application program.” Cline, col. 3:13-16, col. 15:43-49. Cline’s dynamic analysis inserts monitoring code into the application, executes the application using a test harness, and creates a log database. This log database “record[s] system and procedure calls.” Cline, col. 15:53-55, col. 18:61-19:5, Tables 8-10. The database is subsequently used by a ‘post’ program to provide details of “call usage and program coverage.” Cline, col. 3:29-37, col. 15:50-63.

129. Ji describes techniques “detecting and removing computer viruses from file and message transfers between computer networks.” Ji, col. 1:6-13. Ji recognized that the new capabilities provided by the Internet resulted in the need to

scan for viruses at certain network connection points (e.g., gateways nodes). Ji, col. 2:19-44. Ji explains that “processing a file before transmission into the network and [] processing a file before transmission from the network,” provides protection from certain types of viruses. Ji, col. 3:4-16, col. 3:60-63. The gateway node providing scanning and protection may include file transfer (FTP) and/or a mail transfer (SMTP) proxy server. Ji, col. 4:56-5:8. Ji also teaches that its scanning and protection techniques do not need to be located on a gateway but can “be included on a FTP server or a world wide web server for scanning files and messages as they are downloaded from the web.” Ji, col. 5:28-38. After a virus is detected in a received executable file, Ji describes a number of actions that may be taken. Ji, col. 8:6-19.

8. Cline in view of Ji teaches a computer-based method and system for managing Downloadables

130. Cline describes a “digital computer system [that] includes both hardware and software . . . such as [a] central processing unit (CPU), memory, input/output (I/O) ports and peripherals.” Cline, col. 1:12-28. As discussed above, this system is used to test and certify software and application programs that are distributed to different computers (i.e., Downloadables). Cline, col 1:9-11 (the “present invention relates generally to computer software and more particularly to software designed to test application programs”). Cline teaches techniques for

“certifying the portability of software between computer systems.” Cline, col. 2:58-3:5; col. 3:6-21 (“certification tests include a static analysis and a dynamic analysis of the application program . . . [i]f no errors are detected in either the static analysis or the dynamic analysis then the application program is certified.”), col. 2:61-65 (“This invention therefore allows application program developers to produce software which conforms to open standards, thereby greatly increasing the potential market for their application programs.”).

131. Based on the disclosure in Cline, including that Cline is directed to software distribution and software was increasingly being distributed over networks, one of ordinary skill in the art would have understood that Cline teaches a system for managing Downloadables (e.g., analyzing application programs and code for delivery via a network). *See also* ‘494 patent at 2:59-3:8 (stating that “application programs” and “software components” are examples of “Downloadables”), 9:46-52 (same). Additionally, as I explain in detail below, Ji teaches that executable programs and code can be received and/or intercepted over a network and analyzed prior to being delivered to an intended recipient (e.g., client computer). Thus, when these teachings are combined with Cline, it would have been obvious to one of ordinary skill in the art that Cline’s techniques could be used to analyze programs received or intercepted over a network (i.e., incoming Downloadables).

132. Cline also teaches the use of computer software in the form of instruction sequences (i.e., a computer based method). Cline Abstract, col 1:9-11 (the “present invention relates generally to computer software and more particularly to software designed to test application programs”), col. 1:16-18 (the “software portion of the system comprises a sequence of instructions stored in memory which direct and control the operation of the hardware.”), col. 2:66-3:5 (a “method for certifying the portability of software between computer systems”), claims 1, 3, 8 (a “computer implemented method of verifying conformance or non-conformance of an application program to rules that define services which an operating system will provide.”), FIGS. 2, 7, 12, 14.

9. Cline in view of Ji teaches [a receiver] for receiving an incoming Downloadable

133. Cline describes a computer system that includes “input/output (I/O) ports.” Cline col. 1:12-18. One of ordinary skill in the art at the time of the ‘494 patent would have understood that typical computer I/O ports would have included bus connectors (e.g. ISA, EISA, PCI), parallel, and/or serial ports, including ports used to couple the computer system to communication hardware such as network cards, or modems. Such network cards and/or modems were routinely used for receiving data over various networks (*i.e.* receivers for receiving data over networks).

134. As discussed above, Cline teaches systems and methods for testing application programs and verifying their conformance with certain rules and standards. Cline also explains that these certification techniques allow programs to be widely distributed and run on different types of computer systems. Cline, col. 2:58-65 (the “present invention certifies that tested and verified application programs will run on any hardware and operating systems which were designed in conformance with a set of system rules . . . thereby greatly increasing the potential market for [developers’] application programs.”), col. 2:66-3:5 (a “method for certifying the portability of software between computer systems”). In other words, the “portability” of these programs was verified in order to allow them to be distributed to different types of computer systems.

135. Almost since their inception, computer networks have been used to distribute executable code – since it provided more convenience than the alternative (physical distribution). For example, this was frequently done using a modem to connect to an online service provider (e.g., AOL), which provided downloadable programs. As networks became faster, the size of these programs have increased.

136. Thus, in my opinion, one of ordinary skill in the art would have understood that the software and application programs analyzed and verified in Cline could include application programs (i.e., Downloadables) received over a

network (e.g., via the I/O ports). Indeed, the '494 patent acknowledges that such mechanisms for "receiving an incoming Downloadable" were well-known. '494 patent, col. 2:22-44; '194 patent, col. 1:24-57.

137. Ji also teaches "receiving an incoming Downloadable." As discussed above, Ji describes systems for scanning and verifying incoming data, including executable code, at the connection points between networks (e.g., gateways and email relays). Ji, col. 2:13-29 (explaining that, "[w]ith the advent of the Internet and its increased popularity, there are no prior art methods that have been able to successfully scan connections 36 such as those utilized by a gateway node in communicating with other networks."), col. 3:52-62 (describing a "novel gateway node 33 that also performs virus detection for all files being transmitted into or out of a network."). This description mimics the examples discussed in the '494 patent. '494 patent, col. 3:3-21 (stating that gateways and email relays as "devices/processes that are capable of receiving-and-transferring a Downloadable."), col. 5:60-6:7.

138. Ji explains these techniques include scanning files and messages including program code that are received or downloaded over a network. Ji, col. 3:4-16 (describing "a method for processing a file before transmission into the network," which includes "receiving the data transfer command and file name [and] performing virus detection on the file"), Ji, col. 2:30-36 (explaining that the

system “scans program code that is being copied onto the system [and] searches for known patterns of program code used for viruses.”). According to Ji, “the present invention could also be included on a FTP server or a world wide web server for scanning files and messages as they are downloaded from the web.” Ji, col. 5:28-38.

139. In my opinion, one of ordinary skill in the art would have found it obvious to combine the teachings of Cline and Ji. Both Cline and Ji describe techniques for scanning/analyzing executable programs and code (i.e., Downloadables). Cline, col. 2:58-3:21 (“certification tests include a static analysis and a dynamic analysis of the application program”), col. 11:31-37; Ji, col. 2:1-11 (“scans program code that is being copied onto the system”).

140. One of ordinary skill would have understood that signature scanning techniques used static analysis to parse executable code to identify a “virus kernel.” A virus kernel is particular code which, if found is deemed (e.g., by a virus scanner) to be indicative of a known virus. In other words, both Cline and Ji use static techniques (e.g., code parsing) to scan executable code.

141. Based on the teachings in Ji, it would have been obvious to apply Cline’s certification mechanisms (e.g., the SBV and/or DBV analyzers) to “incoming Downloadables” in order to test and verify programs or code received

over a network (e.g., program code received at a gateway or downloaded from the web).

142. In my opinion, one of ordinary skill would have been motivated to combine these teachings for various reasons. For example, one of ordinary skill would have found it useful to verify that incoming Downloadables conform to certain rules before allowing computers on a network (e.g., an Intranet) to download and execute the Downloadables. This technique would have allowed the client computer to avoid potential conformance problems with the incoming program.

143. Also, Ji explicitly states that scanning software at a gateway, rather than on each client machine, is more efficient. Ji, col. 2:12-29. As a result, one of ordinary skill in the art would have been motivated to use such network-based scanning in order to take advantage of certain corporate networks (e.g., Intranets) where computers systems are often similar and security rules are enforced by an “administrator.”

144. For one of ordinary skill in the art, combining the teachings of Cline and Ji would have merely involved combining well-known prior art elements, such as a gateway scanner with a code conformance analyzer using well-known software programming techniques. Such a combination would have and yielded a

predictable result (i.e., a gateway that receives Downloadables and scans them for conformance with rules).

10.Cline in view of Ji teaches [a Downloadable scanner coupled to the receiver for] deriving security profile data for the Downloadable, including a list of suspicious computer operations that may be attempted by the Downloadable

145. Cline describes using both static and dynamic analyses to determine the procedure calls made by executable code (i.e., Downloadables). Cline, Abstract, col. 3:6-36 (“certification tests include a static analysis and a dynamic analysis of the application program”). Cline identifies, verifies and monitors the external calls made by Downloadables, (i.e., functions or operations that are not defined within the executable code itself, such as system calls and library calls). Cline, col. 3:6-21 (“First, a conformance database is developed which includes allowable external calls, such as system calls and procedure calls.”), col. 9:67-10:19 (describing an analyzer that “verifies and validates the system and library calls in the application”), FIGS. 7, 9.

146. In particular, Cline describes a dynamic compatibility verifier (DBV), which addresses certain limitations with static analysis (e.g., difficulties with analyzing inputs that are not generated or resolved until runtime). The DBV is used to perform a dynamic analysis of a program while it is executing using a test harness. Cline, Abstract, col. 3:6-21, col. 15:43-49; *see also* ‘639 provisional, p.

36 (discussing known problems with pure textual/static analysis). DBV “add[s] a small amount of monitoring code to the application program and verif[ies] system and procedure calls and determining program coverage as the application program executes a test program.” Cline, col. 3:29-37, col. 18:42-60 (the “test harness should be as thorough as possible by causing the application program to fully exercise all of its routines and as many as possible if the system and system library calls”).

147. One of ordinary skill would have understood that inserting monitoring code involves some form of textual analysis (i.e., parsing of the code). For example, textual analysis could be used to identify where in the code to place the monitoring routines.

148. During program execution, this “monitoring code can monitor and **record system and procedure calls in a log database.**” Cline, col. 15:50-62 (emphasis added), col. 17:25-48 (“DBV recognizes all system calls during the same pass that it recognizes procedure calls... A system call stub is generated for each detected trap instruction.”). As an example, DBV analysis can be used to “list all user, procedure and system calls (with their parameters) in the order that they are executed.” Cline, col. 19:12-17.

149. In other words, Cline’s DBV (i.e., Downloadable scanner) analyzes Downloadables and lists and records certain calls (i.e., security profile data) that

the Downloadable attempts to invoke. In my opinion, one of ordinary skill in the art would have understood that the data recorded in the log database by the DBV, e.g., system calls, to be a security profile because it lists all of the external calls (i.e., computer operations) attempted by the program using the test harness. As explained by Cline, the goal is to get 100% coverage of all of the system calls in the program. Cline, col. 20:36-37.

150. This security profile data generated by the DBV identifies and lists specific types of calls, such as system calls and library calls, that need to be analyzed and validated (i.e., “suspicious operations.”). Cline explains that the DBV analysis identifies system calls (*i.e.*, suspicious operations) and differentiates them from other types of calls, such as user procedures. Cline, 16:35-34 (“procedure names are categorized into system symbols (which name procedures in the conformance database) or user symbols (which include all other procedures)”), col. 16:30-17:25.

151. Cline also teaches that the DBV uses certain rules to identify and distinguish between system calls and procedure calls. Cline, col. 17:25-32, col. 17:33-36 (“sys_local calls which are recognized by a different trap vector (0x1c1”). This is similar to techniques used by the code scanner described in the ‘194 patent, because Cline teaches that system calls are resolved by traps they use. ‘194 patent, col. 9:24-29 (“code scanner 325 in step 710 resolves a respective

command in the machine code, and in step 715 determines whether the resolved command is suspicious (e.g., whether the command is one of the operations identified in the list described above with reference to FIG. 3”).

152. Cline further explains that these identified and listed system calls may include, various types of system operations, including memory operations, network operations, window management operations, and file operations. Cline, col. 1:19-25 (“system software, often referred to as the ‘operating system’ . . . perform[s] such functions as storing data to memory, inputting and outputting data”), col. 8:27-35 (“if an application program utilizes networking or X-windows the conformance database can be used to ensure the conforming use of those features.”), col. 8:40-54 (discussing the “open() system call” used for accessing files).

153. Thus, it is my opinion that one of ordinary skill in the art would have understood these identified system calls in Cline to be potentially harmful or undesirable calls (i.e., suspicious operations). One of ordinary skill would have recognized that system calls (particularly those discussed above) were typically used by viruses to perform malicious activity. For example, in order to replicate, a virus typically uses a “write” system call to re-write portions of an executable file or a boot file to include the virus itself. These same types of system calls are given as examples of the “suspicious operations” in the ‘494 patent. ‘639 provisional, p.

18, l. 9-13 (DSP data “includes the fundamental computer operations . . . and may include, READs, WRITEs, file management operations, system management operations, memory management operations and CPU allocation operations.”); ‘194 patent, col. 5:58-6:4.

154. One of ordinary skill would have understood that when combined with the teachings of Ji, the DBV analyzers (*i.e.*, Downloadable scanner) would be coupled to the network components that receive the incoming Downloadables (*i.e.*, the receiver). For example, it would have been obvious that the Downloadable scanner and receiver could both be stored in memory, such as RAM and/or connected via a bus on the same computer system (e.g., a gateway or web server as taught by Ji). ‘194 patent, col. 3:23-46, FIG. 3 (explaining that these claimed components may be coupled to the signal bus). Also, because the data representing the Downloadable would need to flow from the receiver to the DBV analyzer for analysis one of ordinary skill would have understood these components would need to be coupled in some manner.

155. In addition to the DBV, it is also my opinion that Cline’s SBV analyzer is another, separate example of a Downloadable scanner that meets this claim limitation. Cline explains that, during the static analysis portion, a “static binary verifier” (SBV) identifies the “basic blocks” of the application program, determines the usage for each basic block, and builds a “graph,” which indicates

program flow between these basic blocks. Cline, col. 10:6-10 (“a graph... is built from the basic blocks by producing a series of pointers which indicate the direction of execution flow between the basic blocks.”).

156. Cline explains that this graph is inverted and used to determine “whether the program is in static conformance with the conformance database” by validating blocks with library and system calls. Cline, col. 12:52-59 (“If a block ends in a system call it is analyzed for conformance with the conformance database [and] [i]f the block ends in an procedure call to a library routine then the procedure call is analyzed.”), col. 11:6-21 (“analyzer CHECK_CALL.C [] verifies and validates the system and library calls in the application... [by] interact[ing] with the conformance database and the graph to determine whether or not register and register values are set up correctly for library and system calls [and] [e]ach path leading to a particular library call or system call is validated independently.”), col. 11:38-43 (describing types of errors or invalid calls that the SBV detects), col. 9:67-10:19, col. 13:44-53, FIGS. 7, 9.

157. The SBV also generates an output indicating the results of its analysis, an example of which is shown in Table 4 (reproduced below).

TABLE 4

SBV Sample Run

```
$ SBV test
Please Wait . . .
processing file gumby
System call to fcntl: 3 args, in__fcntl [00x10414]
  Invalid subtype (arg 'cmd') for library call, value: 0x17
System call to fpathconf: 2 args, in__fpathconf [00x10b34]
  Arg 'fildes' invalid at 0x10b34, value assigned was 0x401f90
  Arg 'name' invalid at 0x10b34, value assigned was 0x0
  Total Valid ? Invalid
Systems Calls      8  4  2  2
Standard features used: BCS
```

158. Based on my understanding of Cline, this output indicates that the SBV identified a total of eight system calls, and determined four to be conforming (“valid”), two to be non-conforming (“invalid”), and two to be unknown (“?”). Cline col. 15:5-41. In other words, Cline’s SBV (i.e., Downloadable scanner) analyzes Downloadables and generates a graph and/or outputs (i.e., security profile data) that identify certain types of calls that the Downloadable may attempt to invoke.

159. Like the DBV, this security profile data generated by the SBV identifies and lists specific types of calls, such as system calls and library calls, that need to be analyzed and validated (i.e., “suspicious operations.”). Cline explains that the SBV identifies certain types of operations (e.g., system calls) within the program using the conformance database. Cline, col. 3:6-21 (“First, a conformance database is developed which includes allowable external calls, such as system calls”), col. 14:43-46 (“call analysis algorithm begins by looking up the call in the

conformance database [and] [i]f the procedure name is not in the database . . . the call analysis terminates.”), col. 8:63-9:3.

160. As I explain above, one of ordinary skill would have understood these identified system calls to be potentially harmful or undesirable calls (i.e., suspicious operations). In fact, these system calls include the same types of operations (e.g., file and memory operations) that are provided as examples of “suspicious operations” in the ‘494 patent. Additionally, as I explain above, one of ordinary skill would have understood that when combined with the teachings of Ji, the SBV (like the DBV) analyzer (*i.e.*, Downloadable scanner) would be coupled to the network components that receive the incoming Downloadables (*i.e.*, the receiver).

11.Cline in view of Ji teaches [a database manager for] storing the Downloadable security profile data in a database

161. As discussed directly above, Cline describes a DBV analyzer, which identifies and generates a list of suspicious operations that may be used by a Downloadable (*i.e.*, Downloadable security profile data). Cline specifically teaches that this security profile data is stored in a “log database.” Cline, col. 15:50-62 (“monitoring code can monitor and **record system and procedure calls in a log database.**”) (emphasis added), col. 17:25-48(“DBV recognizes all system calls during the same pass that it recognizes procedure calls... A system call stub is

generated for each detected trap instruction.”), col. 17:19-24 (“A chain of user procedure stubs are built for use by the POST program.”).

162. Cline also explains that, following the DBV analysis, the log database is used by a post-execution program called POST. Cline, col. 3:29-37, col. 19:33-43. POST uses the results cataloged in the log database to “print a report of call usage and program coverage after the completion of the application program run.” Cline, col. 3:29-37, col. 15:50-62, col. 19:51-20:15. Thus, one of ordinary skill in the art would have understood that this “log database” in Cline was a collection of security profile data that was organized in order to serve (e.g., allow the data to be accessed by) other applications or processes.

163. As I explain above, Cline also describes a SBV analyzer, which creates a “graph” of “basic blocks,” representing the calls that a Downloadable may attempt to invoke, such as system and/or library calls (*i.e.*, suspicious operations). Cline explains that the executable code is scanned to identify these “basic blocks.” Cline, col. 9:67-10:5, col. 11:63-12:3, FIGS. 7, 8. The “graph” is then built by connecting the identified blocks. Cline, col. 10:6-10 (“a graph... is built from the basic blocks by producing a series of pointers which indicate the direction of execution flow between the basic blocks.”), 10:58-65, 12:33-53, 13:3-9 (describing how basic blocks are connected using target and follower pointers), FIG. 9.

164. In my opinion, one of ordinary skill in the art would have also considered this graph to be a form of “database” (*i.e.*, an organized collection of data). As I explain above, Cline teaches how the basic blocks are organized and connected by pointers in the graph. Cline, col. 10:6-10, 10:58-65, 12:33-53, 13:3-9.

165. Like with the DBV, Cline also explains that the graph generated by the SBV analyzer is used by other processes, (e.g., to perform an “external call analysis” to check whether the executable is in conformance with the rules of the conformance database). Cline, col. 13:43-14:57. Thus, this “external call analysis” is a separate program that is served by a database (e.g., the organized basic block graph containing security profile data).

166. Moreover, in my opinion, merely choosing the logical format/organization of how this security profile data is stored in Cline would have been a simple design choice well within the knowledge of ordinary skill in the art at the time of the ‘494 patent. For example, one of ordinary skill would have readily understood that this data could easily be stored in any number of well-known formats, such a plain-file, flat-file database, relational database, raw disk, excel spreadsheet, etc.

167. Additionally, in my opinion, one of ordinary skill in the art would have understood that Cline’s DBV and SBV analyzers (or the overall system)

included components for managing the storage and retrieval of, and access to, the data in these databases (*i.e.*, a database manager).

168. Database managers were well-known at the time of the '494 patent and were routinely used with many different types of databases to manage and/or control the storage and retrieval of data. These database managers were used to prevent or control access to the database and to prevent corruption of the database. Data corruption can occur if there are two or more concurrent processes seeking to write to the database. A database manager would mediate access in order to prevent corruption. Indeed, Cline itself describes the use of such a database manager for managing a database of rules used by the SBV during the static analysis. Cline, col. 10:39-57.

169. Thus, one of ordinary skill in the art would have found it obvious to use a database manager with the types of databases described in Cline, such as the log database and/or graphs (*i.e.*, to manage the storage and retrieval of the Downloadable security profile data). Also, one of ordinary skill in the art would have been motivated to use a database manager to control access to and/or prevent corruption of the security profile data as discussed above. For one of ordinary skill in the art, this would have simply involved combining well-known elements (*i.e.*, databases and database managers) using conventional software programming

techniques to achieve a predictable result (e.g., a log database or graph that was managed by the database manager).

170. One of ordinary skill in the art would have also understood that such a database manager would be coupled to the DBV and/or SBV analyzers (*i.e.*, Downloadable scanners) that generate the security profile data. It would have been obvious that the database manager and Downloadable scanner could be stored in memory, such as RAM and/or connected on the same computer system (e.g., via a bus). This type of colocation is similar to what is described in the '194 patent as coupling. '194 patent, col. 3:23-46, FIG. 3 (explaining that these claimed components may be coupled to the signal bus). In addition, one of ordinary skill in the art would have understood that, because the data representing the Downloadable code would need to flow from the receiver to each of the analyzers for analysis, and then to the database manager for storage in the database, they would be coupled in some manner.

171. Accordingly, for at least the reasons above, it is my opinion that Cline in view of Ji teaches all of the limitations in claims 1 and 10 of the '494 patent.

12.Cline in view of Ji teaches [the database manager] storing a date & time when the Downloadable security profile data was derived in the database

172. As discussed above, Cline describes a 'POST' program, which generates statistics using the security profile data that was derived by the dynamic

analysis and stored in the log database. Cline, col. 18:61-19:5. Cline also teaches that this log database stores a date and time indicating when the Downloadable security profile data was derived by the DBV analyzer. For example, in Table 8 (reproduced below), Cline illustrates an exemplary set of commands that may be used to perform a DBV analysis. The “post” program is then run, which retrieves and displays certain information from the log database. Cline, col 19:6-55 (explaining that the use of the “-a” option shown in Table 8 causes “cumulative statistics to be generated.”), Table 8. In Table 9 (reproduced below), Cline illustrates another set of commands that may be used to run the “post” program a second time, without selecting the options “-uo.” Cline, col. 19:51-55 (explaining that the use of the “-uo” option shown in Table 8 prints the name of unexecuted procedures and causes portable use of the system calls to be reported), Table 9.

| TABLE 8 | TABLE 9 |
|---|---|
| <div style="border: 1px solid black; padding: 2px; margin-bottom: 5px;">Sample Error Free DBV Run - Partial User Coverage</div> <pre> \$ dbv -a dbv foo # instrument dbv producing foo \$ rm -f foo.V foo.P # delete the log files \$ foo dbv bar # run the instrumented program \$ post -uo foo # print the log files Start DBV validation of foo at 02/01/1990 02:19:45 GMT End DBV validation of foo at 02/01/1990 02:20:35 GMT STANDARD FEATURES USED: BCS, OCS </pre> | <div style="border: 1px solid black; padding: 2px; margin-bottom: 5px;">Sample Error Free DBV Run - Full User Coverage</div> <pre> \$foo foo: [-abceprs2] infile [outfile] \$ post foo Start DBV validation of foo at 02/01/1990 02:19:45 GMT End DBV validation of foo at 02/01/1990 02:20:35 GMT STANDARD FEATURES USED: BCS </pre> |
| Cline, Table 8 (annotated) | Cline, Table 9 (annotated) |

173. As can be seen from the highlighted portion of Table 8, running ‘post’ causes dates and times corresponding to the start and end of the previously-performed DBV analysis (i.e., the date & time when the Downloadable security

profile data was derived) to be retrieved and displayed. Also as shown in the highlighted portion of Table 9, running “post” a second time for the DBV analysis results in these same dates and times being displayed. This means that after the DBV generates the security profile data and stores it in the log database, another program (i.e., “post”) can then retrieve and display the date and time when this data was generated by the DBV. Cline, col. 20:65-67.

174. Tables 8 depicts the “post” program being run once following a DBV analysis, while Table 9 depicts a second later run of DBV analysis and second later call to ‘post.’ As can be seen, this second run results in a different analysis and generates different statistics. Table 9 depicts cumulative results of the Table 8 run, this coincides with the use of the –a option discussed above.

175. In my opinion, one of ordinary skill in the art would have understood this disclosure in Cline to mean that these dates and times were stored in the log database (e.g., so they could be subsequently accessed by the POST program). Generally dates and times (e.g., timestamps) are one of many useful, well-known data types that were routinely stored in databases well before the time of the ‘494 patent.

176. Accordingly, for at least the reasons above, it is my opinion that Cline in view of Ji teaches the additional limitations in claims 2 and 11 of the ‘494 patent.

13.Cline in view of Ji teaches that the Downloadable includes program script

177. As discussed above, Cline with Ji teaches receiving and deriving security profile data for a Downloadable. Cline explains that its static and dynamic analysis techniques can be used to derive security profile data for software and application programs. Cline, 3:6-49. Similarly, Ji teaches scanning and analyzing files and messages that are received over a network and include program code.

178. As such, one of ordinary skill would have found it obvious to use these techniques on program scripts. A program script is merely one particular form of executable code that was commonly used at the time of the '494 patent. Indeed, the '494 patent admits that various kinds of program scripts, including scripts received over a network, were well-known and disclosed in a number of prior art references. '494 patent, col. 2:22-27.

179. Ji also teaches that mechanisms for receiving and analyzing Downloadables “could also be included on . . . a world wide web server for scanning files and messages as they are downloaded from the web.” Ji, col. 5:28-38. It was well known in the art that program scripts were often included in files and messages transmitted via the web to a client. *See e.g.*, Apperson, col. 4:9-10 (recognizing a script is one form of executable code: “code might be in the form of textual scripts, byte codes, P-code, or binary object code”). Client-side scripts

such as Javascript, VBscript, and Python were well-known by the time of the '494 patent.

180. In my opinion, it would have been obvious to one of ordinary skill that the systems and methods used to receive and derive security profile data for application programs and program code, as taught by Cline and Ji, could also be used for other types of Downloadables, such as program scripts. For such a person, this would have merely involved applying the same techniques to another well-known form of executable code (e.g., receiving program scripts at a gateway or web server and using static and/or dynamic analysis to identify suspicious operations in the script). One of ordinary skill would have been motivated to do so for various reasons, such as to improve the effectiveness of the verification (e.g., verifying conformance with a particular script interpreter) and virus detection systems taught by Cline and Ji by enabling them to be used with a wider range of Downloadables.

181. Accordingly, for at least the reasons above, it is my opinion that Cline in view of Ji teaches all of the limitations in claims 5 and 14 of the '494 patent.

14.Cline in view of Ji teaches the suspicious computer operations include calls made to an operating system, a file system, a network system, and to memory

182. As discussed above, Cline teaches using both a SBV (static) and DBV (dynamic) analysis to scan a Downloadable and identify and store suspicious

operations that may be attempted by the Downloadable. Cline also explains that these operations include various types of system calls, including calls made to an operating system, file system, network system and memory.

183. For example, Table 8 of Cline (reproduced below) provides some examples of these system calls. One of ordinary skill would have understood that these system procedures shown in Table 8 are calls that are made to an operating system (e.g., DOS or UNIX); Cline, col. 2:3-6.

TABLE 8-continued

Sample Error Free DBV Run - Partial User Coverage

| | |
|-----------------------------------|-------------|
| System procedure coverage factor: | 56.3% |
| User procedure coverage factor: | 92.0% |
| Unexecuted system procedures: | |
| alarm | atoi |
| brk | ecvt |
| execvp | execv |
| execve | execvp |
| fcvt | fprintf |
| kill | fputc |
| longjmp | memchr |
| pause | putc |
| perror | rename |
| setjmp | sigaction |
| segemptyset | sigaddset |
| sigprocmask | sigdelset |
| sigfillset | sigemptyset |
| sigprocmask | sigfillset |
| sigset_t | sigset_t |
| sigsuspend | sigset_t |
| sleep | sigset_t |
| vfprintf | sigset_t |
| Unexecuted user procedures | |
| err | getpathpart |

No portability violations were detected.

184. Table 8 includes the system operations “exec” and “kill,” which are operations used by the operating system to launch and terminate processes. *See also* Cline, col. 7:45, Table 3 (referring to the getpid() system call, which returns the process id of a running program, which is assigned by the operating system).

185. Also shown above, Cline’s analyzers also identify system calls to memory, e.g., the “memchr” operation. Cline, Table 8. Additionally shown in

Table 8 are examples of file system operations: e.g., “fprintf” and “fputc.” *See also* Cline, Tables 1-3, col. 7:46-50, col. 8:12-26, col. 15:8-10 (describing other calls made to the file system, such as “open,” “chdir,” “readlink,” “read,” “fcntl,” and “fpathconf”).

186. Various network operations are also identified using Cline’s teachings. Cline, col. 8:29-32 (“if an application program utilizes networking or X-windows the conformance database can be used to ensure the conforming use of those features”), Table 10 (showing that an invalid “socket” call has been identified), Cline, col. 8:22-26 (describing the “read()” system call, which may be used with both file systems and network systems). As is well known in the art, certain system calls like “read” operate on a file descriptor. A file descriptor is returned by the operating system and may represent either a file or a network socket depending on how it was granted by the OS.

187. Accordingly, for at least the reasons above, it is my opinion that Cline in view of Ji teaches the additional limitations in claims 6 and 15 of the ‘494 patent.

C. Forrest and Ji

188. Forrest discusses “anomaly intrusion detection.” Forrest, p. 2. Anomaly intrusion detection is well-known technique, where it is assumed that the nature of the intrusion is unknown, but that the intrusion will result in behavior

different from that normally seen in the system.” Forrest, p. 2. Because this type of detection relies on detecting behavior deviating from “normal”, it is important to define “normal.” Forrest, p. 1 (“An important prerequisite of such a system is an appropriate definition of self, which is the subject of this paper.”). Forrest “defines normal behavior in terms of short sequences of system calls in a running process.” Forrest, p. 2.

189. Forrest teaches that these system call sequences are stored in a database associated with a particular process (program). Forrest, p. 2 (“The overall idea is to build up a separate database of normal behavior for each process of interest”), p. 3 (“we scan traces of normal behavior and build up a database of characteristic normal patterns (observed sequences of system calls)”). To generate the behavior pattern, the program is run and data is collected using the `strace` utility. Forrest, p. 4. Forrest explains that deviations from the recorded behavior are detected as potential intrusions. Forrest, p. 8 (“If a program enters an unusual error state during an attempted break-in, and if this error condition executes a sequence of system calls that is not already covered by our normal database, we are likely to notice the attack”). Forrest also teaches its techniques can be adopted for specific configurations (user, machine) and then used to detect intrusions as they occur. Forrest, p. 7 (explaining that the system can be “implemented as an on-line system, in which the kernel checked each system call... [and] each site would

generate its own normal database, based on the local software/hardware configuration and usage patterns.”).

190. Forrest’s description of recording “normal behavior” in a database is consistent, with how “suspicious” operations are recognized and listed in a security profile (DSP data) as explained in the ‘494 patent. In the ‘494 patent, the DSP data simply “includes the fundamental computer operations included in each known Downloadable 307, and may include, READs, WRITEs, file management operations, system management operations, memory management operations and CPU allocation operations.” ‘639 provisional, p. 18, l. 9-13, p. 24, l. 19-p. 25, l. 2 (describing loop commands such as “goto”, “while” “if”, “than” or the like as further examples of potentially suspicious commands); ‘194 patent at col. 5:45-6:3, col. 9:20-42.

191. To explain further, most of the time there is nothing inherently “suspicious” about the “fundamental computer operations” described in the ‘494 patent. Instead, another process is used to ultimately determine whether a given action is actually “suspicious” and should not be allowed. In the ‘494 patent, this is the function of the “security policy.” Correspondingly, in Forrest it is deviation from the normal behavior, that is considered “suspicious.” In other words, Forrest’s database represents both a “security profile,” because it was developed

by analyzing the program and a “security policy” because the profile (the valid sequences of normal program behavior) is what is enforced.

192. This allows Forrest to detect different types of deviations from normal behavior. For example, virus infection of a previously profiled program would most likely cause that program to execute a different sequence of system calls. This deviation would then be detected by Forrest as an “intrusion.” An exploit of the profiled running code (e.g., a buffer overrun) that causes new system calls sequences to be executed would also be detectable.

193. Because the database defines “normal behavior,” if it is to be used to detect viruses or intrusions of a previously profiled program, it is important for the program to be virus and intrusion free at the time of the profiling. Alternatively, Forrest’s system could also be used to model virus/intrusion behaviors by capturing the system call sequences for a compromised program.

1. Forrest in view of Ji teaches a computer-based method and system for managing Downloadables

194. Forrest discloses computer-based systems (e.g., “Sun SPARCstations running unpatched versions of SunOS 4.1.1 and 4.1.4.”). Forrest, p. 4. As discussed above, these systems are used to develop a database of normal program behavior of executable programs (e.g., `sendmail` or `lpr`) and to analyze these

programs for subsequent anomalous behavior characterized by Forrest as intrusions. Forrest, p. 1, 3, Abstract.

195. Forrest uses “computer security methods that are based on the way natural immune systems distinguish self from other.” Forrest, p. 1, Abstract. Forrest teaches that there are “two stages to the proposed algorithm. In the first stage, we scan traces of normal behavior and build up a database of characteristic normal patterns (observed sequences of system calls)... In the second stage, we scan new traces that might contain abnormal behavior, looking for patterns not present in the normal database.” Forrest, p. 3.

196. Based on the disclosure in Forrest, as well as the fact that software was increasingly being distributed over networks, one of ordinary skill in the art would have understood that Forrest’s system could be used to manage Downloadables (e.g., application programs and code received via a network). *See also* ‘494 patent at 2:59-3:8 (stating that “application programs” and “software components” are examples of “Downloadables”), 9:46-52 (same). Additionally, as I explain in detail below, Ji teaches that executable programs and code can be received and/or intercepted over a network and analyzed prior to being delivered to an intended recipient (e.g., a client computer). Thus, when these teachings are combined with Forrest, it would have been obvious to one of ordinary skill in the

art that Forrest's techniques could be used to analyze programs received or intercepted over a network (i.e., incoming Downloadables).

2. Forrest in view of Ji teaches [a receiver for] receiving an incoming Downloadable

197. As discussed above, Forrest discloses computer systems and methods that may be implemented on Sun SPARCstations running unpatched versions of SunOS 4.1.1 and 4.1.4". Forrest, p. 3. One of ordinary skill would have understood that SPARCstations would have included or had the ability to host I/O hardware such as network cards, telephone modems, parallel ports, serial ports, which would be capable of receiving data over a network, such as "incoming Downloadables".

198. Forrest also explains that these systems are capable of running applications such as `sendmail`. Forrest, p. 3, Abstract. `Sendmail` was a well-known email transfer utility that supports protocol, such as Simple Mail Transfer Protocol (SMTP). It was also well known to those of ordinary skill in the art that virtually any type of file (including executable programs) could be attached to e-mails. Attaching files and using `sendmail` would then allow a computer system to receive and transfer attachments to a destination, such as a client computer. In fact, an email relay such as computer running `sendmail` is an example given in the '494 patent of "devices/processes that are capable of receiving-and-

transferring a Downloadable.” ‘494 patent, col. 3:3-21, col. 5:60-6:7; *see also* ‘494 patent, col. 2:22-44 and ‘194 patent, col. 1:24-57 (admitting the concept of “receiving a Downloadable” was well-known in the art).

199. Besides `sendmail` (or e-mail attachments generally) there were a number of well-known and conventional techniques for receiving programs from remote sources. For example, a Downloadable could be received via FTP or HTTP protocols using a modem or network interface card (NIC). Also, modem protocols for transferring files were suitable and well-known (e.g., XMODEM, YMODEM, ZMODEM). Thus, it would have been obvious to one of ordinary skill in the art that the application programs analyzed by Forrest’s systems could include programs received over a network (i.e., incoming Downloadables).

200. Also, one of ordinary skill in the art would have found it obvious to combine the teachings of Forrest and Ji. Both Forrest and Ji are directed to scanning/analyzing executable software (i.e., Downloadables). Forrest, p. 2 (discussing “anomaly intrusion detection” as the primary concern of the paper where it assumed that the nature of the intrusion is unknown, but that the intrusion will result in behavior different from that normally seen in the system.); Ji, col. 2:1-11 (“scans program code that is being copied onto the system”).

201. As I explain above, Forrest teaches using a database of sequences of system calls to detect a deviation from normal program behavior, which can be

used to detect changes caused by a virus. For example, a typical virus might try to open an executable file and copy itself. For a given program, this may represent a new system call sequence, which when using Forrest's techniques would indicate that the program has been the subject of an intrusion (in this case viral infection). Forrest, p. 8 ("if code is replaced inside a running program by an intruder, it would likely execute a sequence of system calls not in the normal database").

202. Ji discusses a related concept it calls "behavior detection," which "monitors the computer or system for important operating system functions such as write, erase, format disk, etc." Ji, col. 1:59-63. Ji also explains that certain actions may be anomalous for a given program and that action may be taken when the new behavior is detected. Ji, col. 1:63-2:1 ("When such operations occur, the program prompts the user for input as to whether such an operation is expected. If such an operation is not expected (e.g., the user was not operating any program that employed such a function), the user can abort the operation knowing it was being prompted by a virus program").

203. Based on the related teachings of Ji and Forrest, it would have been obvious to one of ordinary skill for Forrest's "anomaly intrusion detection" system to receive "Downloadables" and to determine/verify and monitor their behavior. As discussed above, Forrest's anomaly intrusion detection techniques could be used to detect the new behavior of a virus infected file. Also, because Forrest relies

on a database of “normal” behavior, it is similar to Ji’s “behavior analysis.” Ji, col. 1:59-2:1. Ji also teaches that its scanning strategy, which is employed on a gateway or web server, is compatible with a number of “substantive” analysis techniques. Ji col. 7:63-65 (teaching that “those skilled in the art will realize that various other virus detection methods may also be used”).

204. One of ordinary skill would have been motivated to combine these teachings for various reasons. For example, one of ordinary skill would be motivated to use these combined teaching to check for viruses by verifying the behavior of incoming “Downloadables,” which may be distributed to one or more computers on a network (e.g., an Intranet). Indeed, as explained in Ji, such centralized scanning is more efficient than individual scanning at client. Ji, col. 2:12-29 (teaching that scanning software on individual machines rather than a gateway can be inefficient).

205. In my opinion, for one of ordinary skill in the art, such a combination would have merely amounted to combining well-known prior art elements (*i.e.*, a gateway scanner with an anomaly intrusion detector) according to well-known software programming techniques in order to yield a predictable result (*i.e.*, a gateway scanner that receives Downloadables and analyzes their behavior).

206. Ji also teaches that its network-based virus detection techniques include receivers for receiving Downloadables (e.g. executable files). Ji, col. 4:18-

55 (“the network link 52 is preferably a network adapter card including a transceiver that is coupled to a cable or line... network link 52 is responsible for sending, receiving, and storing the signals sent over the network... The transfer of data between networks is broken down into the sending and receiving files and messages which in turn are broken down into packets. The methods of the present invention employ a virus detection scheme that is applied to all transfers of messages and files into or out of a network via its gateway node 33.”). One of ordinary skill in the art would have understood that these receivers described in Ji were conventional and well-known components similar to the types of receivers that would have been included on a Sun SPARCstation.

3. Forrest in view of Ji teaches [a Downloadable scanner for] deriving security profile data for the Downloadable, including a list of suspicious computer operations that may be attempted by the Downloadable

207. As discussed above, Forrest teaches determining normal behavior for a program or process. Forrest explains that normal behavior is determined with respect to system call sequences that are made by the program being analyzed. Forrest, p. 2 (“We define normal behavior in terms of short sequences of system calls in a running process”).

208. Forrest focuses on system calls because they represent a risk to the system when executed. Forrest, p. 3 (“System damage is caused by running

programs that execute system calls. Thus, we restrict our attention to system calls in running processes.”). Normal behavior is captured by “tracing normal runs of the program.” Forrest, p. 2, 3 (“we scan traces of normal behavior and build up a database of characteristic normal patterns (observed sequences of system calls)”), p. 4 (describing developing such a database for `sendmail` by sending test messages).

209. This traced normal behavior represents an approximation of how a given program may perform, since the number of sequences a program may take is large and not all paths maybe explored as part of tracing. Forrest p. 3 (explaining that a program, during normal execution, executes subsets of system call sequences and that the complete set of theoretical of sequence is huge and that there is a high probability that any given execution may produce a new sequence). Therefore, any given recorded system call sequence or set of system call sequences represents “operations that may be attempted by the Downloadable.”

210. In my opinion, one of ordinary skill would have understood that such system calls parallel the types of “suspicious operations” given as examples in the ‘494 patent. ‘639 provisional, p. 18, l. 9-13 (DSP data “includes the fundamental computer operations included in each known Downloadable 307, and may include, READs, WRITEs, file management operations, system management operations,

memory management operations and CPU allocation operations.”); ‘194 patent, col. 5:58-6:4, col. 9:24-29.

211. Generally, examples of these well-known system calls include operations such as fork and exec, which are used by the operating system to create new processes. Other system calls include read, write which can used for file or network operations. Forrest explicitly provides examples of system calls that are invoked by sendmail, such as open, read, mmap, getrlimit, and close. Forrest, p. 3.

212. One of ordinary skill would have understood that mmap is memory operation that creates a new mapping in the virtual address space of the calling process , while getrlimit is related to resource allocation.

213. Forrest teaches that these identified sequences of system calls are maintained in a list (i.e., security profile data). Forrest, p. 3.

| call | position 1 | position 2 | position 3 |
|------|------------|------------|------------|
| open | read | mmap | mmap |
| read | mmap | mmap | |
| mmap | mmap | | |

214. This Downloadable security profile is derived by a Downloadable scanner (e.g., the `strace` utility). Forrest, p. 4 (“The `strace` package, version 3.0, was used to gather information on system calls.”).

215. As I discuss above, one of ordinary skill would have understood this Downloadable scanner was coupled to the receiver (e.g., both could be stored in

memory, such as RAM and/or connected together on the same computer system). '194 patent, FIG. 3, col. 3:23-46 (describing couplings to the signal bus). In addition, because data representing the executable program code would flow from the receiver to the `strace` utility for analysis, one of ordinary skill in the art would have understood that these components were coupled in some manner.

4. Forrest in view of Ji teaches [a database manager for] storing the Downloadable security profile data in a database

216. As discussed above, Forrest teaches determining a program's normal behavior by deriving a list of suspicious operations that may be attempted by the program. Forrest also teaches that this behavior information is stored in a database. Forrest, p. 2 ("The overall idea is to build up a separate database of normal behavior for each process of interest. The database will be specific to a particular architecture, software version and configuration, local administrative policies, and usage patterns."), p. 3 ("we scan traces of normal behavior and build up a database of characteristic normal patterns (observed sequences of system calls")

217. Forrest also teaches that this database may serve an application. In Forrest, the database is used to analyze program behavior based upon newly captured traces of a program's execution. These newly captured traces are then used to detect anomalies (e.g., virus infection or intrusion). Forrest, p. 3 ("we scan

new traces that might contain abnormal behavior, looking for patterns not present in the normal database”), p. 2 (“the database can be used to monitor the process’ ongoing behavior.”), see generally Forrest § 4.3.

218. Forrest also explains that this database can then be used to detect abnormal behavior as it occurs by serving “an on-line application,” using a site specific configuration of normal program behavior. Forrest, p. 7 (“Because our measure is easy to compute and is relatively modest in storage requirements, it could be plausibly implemented as an on-line system, in which the kernel checked each system call made by processes running as root. Under this scheme, each site would generate its own normal database, based on the local software/ hardware configuration and usage patterns.”).

219. Forrest clearly discloses saving security profile data (e.g., a database of system call sequences). In my opinion, choosing the particular logical format/organization for how this data is stored in Forrest would have been a simple design choice well within the knowledge of ordinary skill in the art at the time of the ‘494 patent. For example, one of ordinary skill would have readily understood that this data could easily be stored in any number of well-known formats, such a plain-file, flat-file database, relational database, raw disk, excel spreadsheet, etc.

220. Also, when Forrest’s system is used as an on-line application as taught in Ji, one of ordinary skill in the art would have found it obvious to use a database

manager to aggregate and organize the profiles for each of the individual applications being monitored. A database would represent a logical choice for such a function since it would have allowed the monitoring code to access the associated profiles for each monitored process to perform intrusion detection. Additionally, one of ordinary skill in the art would have understood that this type of aggregation would have had additional benefits. For example, aggregating the profiles in a database would have also allowed for easier backup, since this is a function that was typically provided with or easily added to on-line type databases.

221. As discussed above, database managers were well-known at the time of the '494 patent and were routinely used with many different types of databases to manage and/or control the storage and retrieval of data (e.g., to prevent corruption and coordinate access).

222. Thus, in my opinion, one of ordinary skill in the art would have found it obvious to use a database manager to manage the database of system calls (i.e., DSPs) described in Forrest. One of ordinary skill would have been motivated to use a database manager for various reasons, including to control access to and/or prevent corruption of the security profile data as discussed above.

223. One of ordinary skill in the art would have also understood that such a database manager would be coupled to the `strace` utility (i.e., Downloadable scanner) for generating the security profile data. For example, the database

manager and code scanner could be stored in memory, such as RAM and/or connected on the same computer system (e.g., via a bus). As discussed above, this type of colocation is similar to what is described in the '194 patent as coupling. '194 patent, col. 3:23-46, FIG. 3 (explaining that these claimed components may be coupled to the signal bus). In addition, because the data representing the Downloadable program would flow from the receiver to the `strace` utility, and then to the database manager, one of ordinary skill in the art would have understood that these components were coupled in some manner. One of ordinary skill would have also understood that `strace` is invoked using the program to be analyzed as an input parameter.

224. Accordingly, for at least the reasons above, it is my opinion that Forrest in view of Ji teaches all of the limitations in claims 1 and 10 of the '494 patent.

5. Forrest in view of Ji teaches [the database manager] storing a date & time when the Downloadable security profile data was derived in the database

225. As discussed above, Forrest uses the `strace` utility to generate a database of program behavior. Forrest, p. 4. One of ordinary skill in the art would have understood the `strace` includes command line options that causes the date and time to be included in its output (e.g., the '-t' or '-r' options).

226. Also it is my opinion that one of ordinary skill in the art would have found it obvious to use these types of options when generating the program behavior database, since they were documented as part of the manpage for the call. Once these dates and times (timestamps) are generated as part of the `strace` output, it would have been nothing more than a simple design choice to store them in the database along with the system call sequences.

227. Accordingly, for at least the reasons above, it is my opinion that Cline in view of Ji teaches the additional limitations in claims 2 and 11 of the '494 patent.

6. Forrest in view of Ji teaches that the Downloadable includes program script

228. As discussed above, Forrest with Ji teaches receiving and deriving security profile data for a Downloadable. Forrest explains that its analysis techniques can be used to derive security profile data for software and application programs. Cline, 3:6-49. Similarly, Ji teaches scanning and analyzing files and messages that are received over a network and include program code.

229. As such, one of ordinary skill would have found it obvious to use these techniques on program scripts. A program script is merely one particular form of executable code that was commonly used at the time of the '494 patent. Indeed, the '494 patent admits that various kinds of program scripts, including

scripts received over a network, were well-known and disclosed in a number of prior art references. '494 patent, col. 2:22-27.

230. Ji also teaches that mechanisms for receiving and analyzing Downloadables “could also be included on . . . a world wide web server for scanning files and messages as they are downloaded from the web.” Ji, col. 5:28-38. It was well known in the art that program scripts were often included in files and messages transmitted via the web to a client. *See e.g.*, Apperson, col. 4:9-10 (recognizing a script is one form of executable code: “code might be in the form of textual scripts, byte codes, P-code, or binary object code”). Client-side scripts such as Javascript, VBscript, and Python were well-known by the time of the '494 patent.

231. In my opinion, it would have been obvious to one of ordinary skill that the systems and methods used to receive and derive security profile data for application programs and program code, as taught by Forrest and Ji, could also be used for other types of Downloadables, such as program scripts. For such a person, this would have merely involved applying the same techniques to another well-known form of executable code (e.g., receiving program scripts at a gateway or web server and using Forrest’s analysis to identify suspicious operations in the script). One of ordinary skill would have been motivated to do so for various reasons, such as to improve the effectiveness of the verification (e.g., verifying

conformance with a particular script interpreter) and virus detection systems taught by Forrest and Ji by enabling them to be used with a wider range of Downloadables.

232. Accordingly, for at least the reasons above, it is my opinion that Forrest in view of Ji teaches all of the limitations in claims 5 and 14 of the '494 patent.

7. Forrest in view of Ji teaches that the suspicious computer operations include calls made to an operating system, a file system, a network system, and to memory

233. As discussed above, Forrest teaches that the Downloadable security profile is derived by a Downloadable scanner (e.g., `strace`). Forrest, p. 4 (“The `strace` package, version 3.0, was used to gather information on system calls.”).

234. In my opinion, one of ordinary skill would have understood that `strace` would have the ability to capture all system calls. This naturally would include “calls made to an operating system, a file system, a network system, and to memory.” For example, one of ordinary skill in the art would have understood that in general, system calls are operating system operations (e.g., DOS or UNIX operations). In addition, `strace` would capture operating system operations related to processes, such as “exec.” ‘194 patent, col. 5:66-6:3.

235. Forrest also discusses system calls associated with filesystem operations (e.g., `open`, `read`, `close`). Forrest, p. 3. Forrest also discusses system

calls associated with network operations (e.g., read, close). Forrest, p. 3. As discussed above, certain calls such as read, write, and close operate on file descriptors, which can be associated with a network or a file. As also discussed above, Forrest refers to system calls that are associated with memory (e.g., mmap). Forrest, p. 3.


236. Accordingly, for at least the reasons above, it is my opinion that Forrest in view of Ji teaches all of the limitations in claims 6 and 15 of the '494 patent.

Conclusion

In signing this declaration, I recognize that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the 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.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further 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.

Executed on the 10th day of September, 2015.



Jack W. Davidson