UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CROWDSTRIKE, INC.,

Petitioner

v.

SKYSONG INNOVATIONS, LLC,

Patent Owner

Inter Partes Review Case No. IPR2025-01400

U.S. Patent No. 10,574,721

DECLARATION OF EREZ ZADOK, Ph.D.

TABLE OF CONTENTS

I. EDUCATIONAL BACKGROUND AND QUALIFICATIONS 16

II. METHODOLOGY AND MATERIALS CONSIDERED 27

III. LEGAL FRAMEWORK 33

IV. OPINION 40

A. THE '721 PATENT 40
B. SUMMARY OF THE '721 PATENT'S PROSECUTION HISTORY 43
C. CLAIM CONSTRUCTION 44
D. LEVEL OF A PERSON OF ORDINARY SKILL IN THE ART 45

V. BACKGROUND OF THE INVENTION AND PRIOR ART - LINUX 46

VI. BACKGROUND OF THE TECHNOLOGY 56

A. COMPUTERS AND SERVERS 56
B. WEB BROWSERS, BROWSER EXTENSIONS, AND BROWSER PLUGINS 59
C. SENSITIVE WEB CONTENT AND WEB BROWSER SECURITY 61
D. DOMAIN NAME SYSTEM (DNS) 66
E. DAEMONS 69
F. SYSTEM TRACE 72
G. NATIVE APPLICATIONS 73
H. APPLICATION PROGRAMMING INTERFACE (API) 73
I. LINUX OPERATING SYSTEM 74
J. C PROGRAMMING LANGUAGE 75

VII. OPINION: GROUND 1 77

A. OBVIOUSNESS OVER FAGES, KORYCKI, JAI, AND HAYDEN 77
1. Overview of Fages 77
2. Overview of Korycki 80
3. Overview of Jai 84
4. Overview of Hayden 86
5. Claim 1(Pre): "A system, comprising: a processor, operable to:" 87
a. Korycki Teaches a Processor 91

b. Motivation to Combine Fages with Korycki (Including a Processor in a Server Computer) 93
6. Claim 1(a): "receive data defining a first web content class and a second web content class, the first web content class defining a first plurality of Uniform Resource Locators (URLs), and the second web content class defining a second plurality of URLs different than the first plurality of URLs, the second web content class further being associated with sensitive information presenting a security risk;" 94
a. "receive data defining a first web content class and a second web content class" 94
b. "the second web content class defining a second plurality of URLs different than the first plurality of URLs" 95
c. "the second web content class further being associated with sensitive information presenting a security risk" 99
d. "first web content class defining a first plurality of Uniform Resource Locators (URLs)" 101
7. Claim 1(b): "generate a first browser instance, the first browser instance defining a browser instance type predetermined to be associated with the first web content class;" 104
a. '721 Patent and '721 FH "browser instance type" 104
b. Fages's "first browser instance" and "browser instance type" 106
8. Claim 1(c): "intercept a request to access web content from the first browser instance, the request defining a URL of the second plurality of URLs such that the web content is associated with the second web content class;" ... 115
9. Claim 1(d): "confirm that the URL is not being accessed by another separate already-running browser instance associated with the second web content class, by" 119
a. Fages's Teachings 119
b. Jai's URL-Comparison Technique 120
c. Hayden's Teachings – "*another separate already-running browser instance*" 123
d. The Fages-Korycki-Jai-Hayden Combination Teaches "confirm that the URL is not being

accessed by another separate already-running browser instance…” 126
e. Motivation to Combine Fages with Jai and Hayden (Browser Instance Detection and Active URL Comparison) 131
10. Claim 1(d)(i): “implementing a daemon outside the first browser instance that tracks active browser instances and launches a new browser in order to manage web access according to the data defining the first web content class and the second web content class,” 140
a. Fages’s Teachings 140
b. Korycki’s Teachings 141
c. Motivation to Combine Fages-Jai-Hayden with Korycki (Utilizing a Daemon was Advantageous and Well Known) 145
11. Claim 1(d)(ii): “receiving the request via a browser extension associated with the first browser instance,” 151
a. Korycki’s Browser Extension 151
b. Motivation to Combine Fages-Jai-Hayden with Korycki (Browser Extension).....155
12. Claim 1(d)(iii): “forwarding the request from the browser extension to the daemon using a native application,” 161
a. *Korycki’s* Teachings.....161
b. Motivation to Combine Fages-Jai-Hayden with Korycki (Native Application Provides Communication).....165
13. Claim 1(d)(iv): “confirming, via the daemon that the web content is not associated with the first web content class and that another browser instance is not already launched and available to access the web content; and” 169
14. Claim 1(e): “launch a new browser for accessing the web content associated with the second web content class to provide access to the URL of the second plurality of URLs via the new browser.” 171
15. Claim 2: “The system of claim 1, wherein the new browser instance is launched automatically for display, and the web content is associated with a web server.” 174
VIII. OPINION: GROUND 2 176

A. OBVIOUSNESS OVER FAGES, KORYCKI, JAI, HAYDEN, AND RUSSELLO 176
1. Overview of Russello 176
2. Claim 3: "The system of claim 2, wherein the processor is operable to execute an API call configured to intercept a system call to the web server, the system call to the web server intended to provide access of the URL to the first browser instance." 177
a. '721 Patent's API 177
b. Russello's API Call 177
c. Motivation to Combine Fages-Korycki-Jai-Hayden with Russello (ptrace and API) 181
3. Claim 4(Pre): "The system of claim 3, further comprising:" 185
4. Claim 4(a): "wherein the processor is operable to pause operations associated with the first browser instance upon interception of the system call; and" 185
a. Russello's Pausing Operations 186
b. Motivation to Combine Fages-Korycki-Jai-Hayden with Russello (Pausing Operation Benefits) 187
5. Claim 4(b): "wherein the API controls system calls made by other programs and other browser instances." 188
a. Russello's API Controls System Calls 188
b. Motivation to Combine Fages-Korycki-Jai-Hayden with Russello (API Controls System Calls from Other Programs and Browser Instances) 190
6. Claim 5: "The system of claim 1, wherein the processor is operable to implement a tracer to determine whether or not a requested network object is allowed to be accessed by the first browser instance." 192
a. '721 Patent's Tracer 192
b. Russello's Tracer (Process Monitor Using ptrace) 193
c. Motivation to Combine Fages-Korycki-Jai-Hayden with Russello (Tracer Tracking System Calls) 194

IX. OPINION: GROUND 3 196

A. OBVIOUSNESS OVER FAGES, KORYCKI, JAI-HAYDEN, O'NIELL, AND CARDONA 196
1. Overview of O'Niell 196
2. Overview of Cardona 197

3. *Claim 6: The system of claim 1, wherein the daemon comprises a multi-threaded C program that uses Unix domain sockets to communicate with the browser extension and other extensions.* *198*
   a. O'Niell's C program Daemon Using Unix Domain Sockets 199
   b. Cardona's Multithreaded Daemon 200
   c. Motivation to Combine Fages-Korycki-Jai-Hayden with O'Niell and Cardona (C Programming and Unix Domain Sockets) 201
   d. O'Niell Motivation.....201
   e. Cardona Motivation.....205
4. *Claim 7(Pre): "A method for facilitating different categories of content to be displayed in different independent browser instances, the method comprising:"* ... *208*
5. *Claim 7(a): "utilizing a processor in communication with a non-transitory tangible storage medium storing instructions that are executed by the processor to perform operations comprising:"* *208*
6. *Claim 7(a)(i): "accessing data defining rules for delegating web access associated with a first content class to a first browser type and delegating web access associated with a second content class to a second browser type;"* *208*
7. *Claim 7(a)(ii): "identifying a request to access an IP address by a first browser instance, the first browser instance being of the first browser type;"* *209*
8. *Claim 7(a)(iii): "determining that the IP address is associated with the second content class such that the IP address should not be accessed by the first browser instance and confirming that a separate browser instance is not already in use and available to access the request, by:"* *209*
9. *Claim 7(a)(iii)(1): "implementing a daemon, the daemon comprising a multi-threaded program executed outside the first browser instance that tracks active browser instances and launches a new browser instance in order to manage according to the rules,"* *209*
10. *Claim 7(a)(iii)(2): "receiving the request via a browser extension associated with the first browser instance,"* *210*

11. *Claim 7(a)(iii)(3): "forwarding the request from the browser extension to the daemon using domain sockets; and"* *210*
12. *Claim 7(a)(iv): "generating a second browser instance to access the IP address, the second browser instance being of the second browser type."* *210*
13. *Claim 8: The method of claim 7, wherein the second browser instance is launched automatically to a client device for display.* *210*
14. *Claim 12(Pre): "A non-transitory computer readable medium encoded with a set of instructions for facilitating different categories of content to be displayed in different independent browser instances, the set of instructions, executable by a processor, comprising:"* *211*
15. *Claim 12(a): "accessing data associating a first web content class with a first browser type and a second web content class with a second browser type;"* *211*
16. *Claim 12(b): "accessing a request to access web content via a first browser instance, the first browser instance being of the first browser type; and"* *211*
17. *Claim 12(c): "comparing the web content to the first web content class and the second web content class to determine whether the web content is accessible to the first browser instance, by"* *211*
18. *Claim 12(c)(i): "implementing a daemon, the daemon comprising a multi-threaded program executed outside the first browser instance that tracks active browser instances and launches a new browser instance in order to manage according to the rules,"* *211*
19. *Claim 12(c)(ii): "receiving the request via a browser extension associated with the first browser instance,"* *212*
20. *Claim 12(c)(iii): "forwarding the request from the browser extension to the daemon using domain sockets."* *212*
21. *Claim 13(Pre): "The non-transitory computer readable medium of claim 12, further comprising:"* *212*
22. *Claim 13(a): "generating a second browser instance to access the web content,"* *212*
23. *Claim 13(b): "wherein the second browser instance is launched automatically to a client device for display."* *212*

24. Claim 16: The non-transitory computer readable medium of claim 12, wherein the second web content class is associated with sensitive information indicating that accessing the web content by the first browser instance presents a security risk. 212
25. Claim 17: The non-transitory computer readable medium of claim 12, wherein the first web content class is associated with a first plurality of URLs and the second web content class is associated with a second plurality of URLs, the first plurality of URLs being different from the second plurality of URLs. 212
26. Claim 18: The non-transitory computer readable medium of claim 12, wherein the data further defines rules limiting web access by the first browser instance to URLs defined by the first web content class. 213

X. OPINION: GROUND 4 213

A. OBVIOUSNESS OVER FAGES, KORYCKI, JAI-HAYDEN, O'NIELL, CARDONA, AND RUSSELLO.....213
1. Claim 9: The method of claim 7, further comprising: executing an API call configured to intercept a system call to a web server associated with the IP address, the system call initiated by the first browser instance..... 213
2. Claim 10: The method of claim 7, further comprising: executing a tracer to determine whether or not a requested network object is allowed to be accessed by the first browser instance. 213
3. Claim 14(Pre): "The non-transitory computer readable medium of claim 12, further comprising:" 213
4. Claim 14(a): "executing an API call configured to intercept a system call to a web server," 213
5. Claim 14(b): "the system call associated with the first browser instance." 214
6. Claim 15: The non-transitory computer readable medium of claim 12, further comprising: executing a tracer to determine whether or not the first browser instance is authorized to access a requested network. 214

XI. OPINION: GROUND 5 214

A. OBVIOUSNESS OVER FAGES, KORYCKI, JAI, HAYDEN, O'NIELL, CARDONA, AND TAZUMA 214
1. Overview of Tazuma 214
2. Claim 11(Pre): "The method of claim 7, further comprising:" 215
3. Claim 11(a): "generating a domain name system (DNS) query to resolve the IP address;" 215
a. Tazuma's DNS Query 216
b. Motivation to Combine Fages-Korycki-Jai-Hayden-O'Niell-Cardona with Tazuma (DNS Query) 216
4. Claim 11(b): "determining that the IP address is associated with the second content class; and" 218
5. Claim 11(c): "issuing a DNS response, the IP address being rewritten in the DNS response to prevent the first browser instance from accessing the IP address and information of the second content class; and" 219
a. '721 Patent DNS Response 219
b. Tazuma's DNS Response 219
c. Motivation to Combine Fages-Korycki-Jai-Hayden-O'Niell-Cardona with Tazuma (DNS Redirection) 220
6. Claim 11(d): "returning the IP address as rewritten to the first browser instance." 223
a. Tazuma's Returning a Rewritten IP Address 223
b. Motivation to Combine Fages-Korycki-Jai-Hayden-O'Niell-Cardona with Tazuma (DNS Redirection) 225

XII. CONCLUSION 226

Challenged Claims

Claim Designation	Claim Language
Claim 1(Pre)	A system, comprising: a processor, operable to:
Claim 1(a)	receive data defining a first web content class and a second web content class, the first web content class defining a first plurality of Uniform Resource Locators (URLs), and the second web content class defining a second plurality of URLs different than the first plurality of URLs, the second web content class further being associated with sensitive information presenting a security risk;
Claim 1(b)	generate a first browser instance, the first browser instance defining a browser instance type predetermined to be associated with the first web content class;
Claim 1(c)	intercept a request to access web content from the first browser instance, the request defining a URL of the second plurality of URLs such that the web content is associated with the second web content class;
Claim 1(d)	confirm that the URL is not being accessed by another separate already-running browser instance associated with the second web content class, by
Claim 1(d)(i)	implementing a daemon outside the first browser instance that tracks active browser instances and launches a new browser in order to manage web access according to the data defining the first web content class and the second web content class,
Claim 1(d)(ii)	receiving the request via a browser extension associated with the first browser instance,
Claim 1(d)(iii)	forwarding the request from the browser extension to the daemon using a native application,

Claim Designation	Claim Language
Claim 1(d)(iv)	confirming, via the daemon that the web content is not associated with the first web content class and that another browser instance is not already launched and available to access the web content; and
Claim 1(e)	launch a new browser for accessing the web content associated with the second web content class to provide access to the URL of the second plurality of URLs via the new browser.
Claim 2	The system of claim 1, wherein the new browser instance is launched automatically for display, and the web content is associated with a web server.
Claim 3	The system of claim 2, wherein the processor is operable to execute an API call configured to intercept a system call to the web server, the system call to the web server intended to provide access of the URL to the first browser instance.
Claim 4(Pre)	The system of claim 3, further comprising:
Claim 4(a)	wherein the processor is operable to pause operations associated with the first browser instance upon interception of the system call; and
Claim 4(b)	wherein the API controls system calls made by other programs and other browser instances.
Claim 5	The system of claim 1, wherein the processor is operable to implement a tracer to determine whether or not a requested network object is allowed to be accessed by the first browser instance.
Claim 6	The system of claim 1, wherein the daemon comprises a multi-threaded C program that uses Unix domain sockets to communicate with the browser extension and other extensions
Claim 7(Pre)	A method for facilitating different categories of content to be displayed in different independent browser instances, the method comprising:

Claim Designation	Claim Language
Claim 7(a)	utilizing a processor in communication with a non-transitory tangible storage medium storing instructions that are executed by the processor to perform operations comprising:
Claim 7(a)(i)	accessing data defining rules for delegating web access associated with a first content class to a first browser type and delegating web access associated with a second content class to a second browser type;
Claim 7(a)(ii)	identifying a request to access an IP address by a first browser instance, the first browser instance being of the first browser type;
Claim 7(a)(iii)	determining that the IP address is associated with the second content class such that the IP address should not be accessed by the first browser instance and confirming that a separate browser instance is not already in use and available to access the request, by:
Claim 7(a)(iii)(1)	implementing a daemon, the daemon comprising a multi-threaded program executed outside the first browser instance that tracks active browser instances and launches a new browser instance in order to manage according to the rules,
Claim 7(a)(iii)(2)	receiving the request via a browser extension associated with the first browser instance,
Claim 7(a)(iii)(3)	forwarding the request from the browser extension to the daemon using domain sockets; and
Claim 7(a)(iv)	generating a second browser instance to access the IP address, the second browser instance being of the second browser type.
Claim 8	The method of claim 7, wherein the second browser instance is launched automatically to a client device for display.

Claim Designation	Claim Language
Claim 9	The method of claim 7, further comprising: executing an API call configured to intercept a system call to a web server associated with the IP address, the system call initiated by the first browser instance.
Claim 10	The method of claim 7, further comprising: executing a tracer to determine whether or not a requested network object is allowed to be accessed by the first browser instance.
Claim 11(Pre)	The method of claim 7, further comprising:
Claim 11(a)	generating a domain name system (DNS) query to resolve the IP address;
Claim 11(b)	determining that the IP address is associated with the second content class; and
Claim 11(c)	issuing a DNS response, the IP address being rewritten in the DNS response to prevent the first browser instance from accessing the IP address and information of the second content class; and
Claim 11(d)	returning the IP address as rewritten to the first browser instance.
Claim 12(Pre)	A non-transitory computer readable medium encoded with a set of instructions for facilitating different categories of content to be displayed in different independent browser instances, the set of instructions, executable by a processor, comprising:
Claim 12(a)	accessing data associating a first web content class with a first browser type and a second web content class with a second browser type;
Claim 12(b)	accessing a request to access web content via a first browser instance, the first browser instance being of the first browser type; and

Claim Designation	Claim Language
Claim 12(c)	comparing the web content to the first web content class and the second web content class to determine whether the web content is accessible to the first browser instance, by
Claim 12(c)(i)	implementing a daemon, the daemon comprising a multi-threaded program executed outside the first browser instance that tracks active browser instances and launches a new browser instance in order to manage according to the rules,
Claim 12(c)(ii)	receiving the request via a browser extension associated with the first browser instance,
Claim 12(c)(iii)	forwarding the request from the browser extension to the daemon using domain sockets.
Claim 13(Pre)	The non-transitory computer readable medium of claim 12, further comprising:
Claim 13(a)	generating a second browser instance to access the web content,
Claim 13(b)	wherein the second browser instance is launched automatically to a client device for display.
Claim 14(Pre)	The non-transitory computer readable medium of claim 12, further comprising:
Claim 14(a)	executing an API call configured to intercept a system call to a web server,
Claim 14(b)	the system call associated with the first browser instance.
Claim 15	The non-transitory computer readable medium of claim 12, further comprising: executing a tracer to determine whether or not the first browser instance is authorized to access a requested network.

Claim Designation	Claim Language
Claim 16	The non-transitory computer readable medium of claim 12, wherein the second web content class is associated with sensitive information indicating that accessing the web content by the first browser instance presents a security risk.
Claim 17	The non-transitory computer readable medium of claim 12, wherein the first web content class is associated with a first plurality of URLs and the second web content class is associated with a second plurality of URLs, the first plurality of URLs being different from the second plurality of URLs.
Claim 18	The non-transitory computer readable medium of claim 12, wherein the data further defines rules limiting web access by the first browser instance to URLs defined by the first web content class.

I, Erez Zadok, Ph.D., declare as follows:

1. I am over the age of 21 and am competent to make this declaration. I make this declaration based on facts and matters within my own knowledge and on information provided to me by others, and, if called as a witness, I could and would competently testify to the matters set forth herein.

I. EDUCATIONAL BACKGROUND AND QUALIFICATIONS

2. My *curriculum vitae*, which includes a more detailed summary of my background, experience and publications, is provided as EX1011.

3. I am a Professor in the Computer Science Department at Stony Brook University (part of the State University of New York ("SUNY") system). I direct the File-systems and Storage Lab (FSL) at Stony Brook's Computer Science Department. My research interests include file systems and storage systems, operating systems, transactional systems including database technologies, information technology and system administration, security/privacy and information assurance, networking, energy efficiency, performance and benchmarking, virtualization, cloud systems, compilers, applied machine learning, and software engineering.

4. I studied at a professional high school in Israel, focusing on electrical engineering ("EE"), and graduated in 1982. I spent one more year at the high school's college division, receiving a special Certified Technician's degree in EE. I

then went on to serve in the Israeli Defense Forces for three years (1983–1986). I received my Bachelor of Science degree in computer science ("CS") in 1991, my Master's degree in CS in 1994, and my PhD in CS in 2001—all from Columbia University in New York.

5. When I began my undergraduate studies at Columbia University, I also started working as a student assistant in the various campus-wide computer labs, eventually becoming an assistant to the head labs manager, who was managing all public computer labs on campus. During that time, I also became more involved with research within the CS Department at Columbia University, conducting research on operating systems, file and storage systems, distributed and networked systems, security, and other topics. I also assisted the CS department's computer administrators in managing the department's computers, which included storage, IT, networking, and cyber-security related duties.

6. In 1991, I joined Columbia University's CS department as a full-time systems administrator, studying towards my MS degree part-time. My MS thesis topic is related to file system reliability, fault tolerance, replication, and failover in mobile networked storage systems using file virtualization. My main duties as a systems administrator involved installing, configuring, and managing many networked servers, proxies, and desktops running several operating systems, as well as network devices setup; this included many software and hardware upgrades,

device upgrades, and BIOS firmware/chipset updates/upgrades. My duties also included ensuring reliable, secure, authenticated access to networked systems/storage and licensed software, as well as software updates, security and bug fixes. Examples of servers and their protocols included email (SMTP), file transfer (FTP), domain names (DNS), network file systems (NFS), network news systems (NNTP), and Web (HTTP).

7. In 1994, I left my systems administrator position to pursue my doctoral studies at Columbia University. My PhD thesis topic was on versatile file system development using stackable (virtualized) file systems, with examples in the fields of security and encryption, efficiency, reliability, and failover. I continued to work part-time as a systems administrator at the CS department, and eventually I was asked to serve as manager to the entire information technology ("IT") staff. From 1991 to 2001, I was also a member of the faculty-level Facilities Committee that oversaw all IT operations at the CS department.

8. As part of my PhD studies at Columbia, I collaborated on projects to develop advanced AI-like techniques to detect previously unknown viruses (a.k.a. "zero-day malware"), using data mining and rule-based detection. This work led to several highly cited papers (over 1,700 citations for one of the papers alone) and two patents. I also became a Teaching Assistant ("TA") for a first-ever Computer

Security course given at Columbia University's CS department with Dr. Matt Blaze as instructor.

9. From 1990 to 1998, I consulted for SOS Corporation and HydraWEB Technologies as a systems administrator and programmer, managing data storage use and backup/restore duties, databases, web servers, as well as information assurance and cyber-security (*e.g.*, malware protection, software licensing). From 1994 to 2000, I led projects at HydraWEB Technologies and then became the Director of Software Development—overseeing the development of several products and appliances such as stateful firewalls and HTTP load-balancers, utilizing network-virtualization and high-availability techniques. From 2009 to 2019, I have consulted for Packet General Networks, a startup specializing in secure, virtualized, network storage and applications' data security in the cloud.

10. In 2001, I joined the faculty of Stony Brook University, a position I have held since that time. In 2002, I joined the Operations Committee, which oversees the IT operations of the CS department at Stony Brook University. From 2006 to 2010, I was the Director of IT Operations of the CS department. My day-to-day duties included setting policies regarding computing, hiring and training new staff, assisting any staff with topics of my specialty, defining requirements for new software/hardware, and purchasing. From 2010 to 2015, I served as the Co-Chair to the Operations Committee. From 2016 to 2019, I oversaw the IT Operations as the

Chair of the Operations Committee. A significant component of these duties included defining and helping implement policies for data management, to ensure the security of users and their data, and data reliability and availability, while minimizing the inconvenience and performance impact to users. I personally helped setup and maintain an initial virtual-host infrastructure in the department. Since late 2019, I've been a member of the department's Executive Committee that also oversees all IT operations.

11. In 2017, I became the department's Graduate Academic Adviser, advising all Master students (over 400 annually on average) and many other graduate students on an assortment of academic matters. In 2024, I took over as the department's Graduate Program Director, overseeing the entire graduate CS program (700-800 students annually on average).

12. Since 2001, I have personally configured and managed my own research lab's network. This includes setting up and configuring multiple storage systems (*e.g.*, NFS, CIFS/SMB, NAS), virtual and physical environments, applications such as database (*e.g.*, MySQL, PostgreSQL), Web servers (*e.g.*, Apache), and mail servers; user access control (*e.g.*, NIS, LDAP), backups and restores, snapshot policies, and more. I have personally installed, configured, changed, replaced parts, and upgraded components in numerous devices including mobile devices, laptops, desktops, and servers, both physical and virtual.

13. Since 1995, I have taught courses on operating systems, storage and file systems, advanced systems programming in Unix/C, systems administration, data structures, data/software security, and more. My courses often use storage, file systems, distributed systems, and system/network security as key teaching principles and practical examples for assignments and projects. I have taught these concepts and techniques to my students, both to my direct advisees as well as in my courses. For example, in my graduate Operating Systems course, I often cover Linux's kernel mechanisms to protect users, applications, and data files, virtual file systems, as well as distributed storage systems (*e.g.*, NFS). And in the System Administration undergraduate course, I covered many topics such as networking, storage, backups, and configuring complex applications such as mail, web, and database servers.

14. My research often investigates computer systems from many angles: security, efficiency, energy use, scalability, reliability, portability, survivability, usability, ease-of-use, versatility, flexibility, and more. My research gives special attention to balancing five often-conflicting aspects of computer systems: performance, reliability, energy use, security, and ease-of-use.

15. Since joining Stony Brook University in 2001, my group in the File-systems and Storage Lab ("FSL") has developed many file systems and operating system extensions; examples include a highly-secure cryptographic file system, a portable copy-on-write ("COW") versioning file system, a tracing file system useful

to detect intrusions, a replaying file system useful for forensics, a snapshotting and sandboxing file system, a namespace unification file system (that uses stackable, virtualized, file-based COW), an anti-virus file system, an integrity-checking file system, a load balancing and replication/mirroring file system, network file system extensions for security and performance, distributed secure cloud-based storage systems, transactional key-value stores and file systems, OS-level embedded databases, a compiler to convert user-level C code to in-kernel efficient yet safe code, GCC plugins, stackable file system templates, and a Web-based backup system. Many of these projects used one form of virtualization or another (storage, network, host, etc.). I continue to maintain and release newer versions of some of these file systems and software.

16. I have published over 120 refereed publications (in ACM, IEEE, USENIX, and more). To date, my publications have been cited more than 10,000 times (as per Google Scholar as of August 25, 2025). My papers cover a wide range of related technologies such as file systems, storage systems, transactional systems, security, clouds and virtualization, performance benchmarking and optimization, energy efficiency, system administration, web systems, and more. I also published a book titled "Linux NFS and Automounter Administration" (Sybex, 2001), covering systems administration topics related to network storage and data security.

17. Some of my research has led to public software releases that have been used worldwide. I have publicly maintained the Amd Berkeley Automounter in a package called "am-utils" since 1992; this software helps administrators manage the multitude of file system mounts on dozens of different Unix systems, especially helping to automate access to multiple NFS/NAS storage volumes. Since 1997, I have maintained and released several stackable (virtualized) file system software projects for Linux, FreeBSD, and/or Sun Solaris, in a package called FiST. One of my stackable file system encryption projects, called Cryptfs, became the basis for IBM's public release of eCryptfs, now part of Linux. Packet General Networks, for whom I have provided consulting services between 2009 and 2019, licensed another encryption file system called Ncryptfs. Another popular file system released in 2003, called Unionfs, offers virtual namespace unification, transparent shadow copying (a.k.a. copy-on-write or COW), file system snapshotting (*e.g.*, useful for forensics and disaster recovery), and the ability to save disk space by sharing a read-only copy of data across several computers, among other features.

18. My research and teaching make extensive use of data security features. For example, each time I taught the graduate operating system course, the first homework assignment includes the creation of a new system call that performs new or added functionality, often for encrypting a file or verifying its integrity; many of my other assignments cover topics of user/process access control, anti-virus filtering,

and more. Since 2001, over 1,000 graduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

19. Moreover, in an undergraduate course titled "Advanced Systems Programming in Unix/C," I cover many topics of system security and vulnerabilities, such as the structure of UNIX processes, and memory segments such as the heap and stack. This course covers details of several hundred Linux system calls. Often, the first assignment for this course is to develop a tool to encrypt/decrypt files using advanced ciphers, use digital signatures to certify the cipher keys used, and reliably recover files in case of failures. Since 2001, several hundred undergraduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

20. In another undergraduate course, System Administration, I taught network configuration, security, and storage configuration and reliability. In a special topics course on Storage Systems, I covered many topics such as data deduplication, RAID, transactional storage, storage hardware including modern Flash based ones, virtual storage, backup/restore, snapshots and continuous data protection ("CDP"), NAS and SAN, and NFS.

21. Overall, in addition to the aforementioned experience, my technical experience relevant to the '721 Patent at the time of the alleged invention included the following: I studied, researched, developed code, published, taught,

administered, and/or released software for the following: managed, modified, and used multiple web servers and browsers; used process confinement techniques such as chroot, containers, and Linux cgroups; used, developed, and modified tracing/interception technologies with ptrace, ktrace, dtrace, strace, ltrace, and packet tracers (including inside the Linux/Unix kernel); wrote inter-process communication code using IPC, TCP sockets, and Unix/Linux domain sockets; wrote/managed several daemons including amd, hlfsd, and nfsd; developed firewall and system/network security rules and implementations; developed web browser CGI scripts, modified Apache plugins; used and modified process monitors such as "nanny"; developed and used internal and external process-monitoring techniques; and managed and used redirection techniques using DNS, HTTP, and packet firewalls.

22. My research has been supported by many federal and state grants as well as industry awards, including an NSF CAREER award, two IBM Faculty awards, two NetApp Faculty awards, a Western Digital award, a Facebook award, several Dell-EMC awards, and several equipment gifts. I received the 2008 SUNY Chancellor's Excellence in Teaching award, and the 2022 SUNY Chancellor's Award for Excellence in Scholarship and Creative Activities (both awards can be given only once in a lifetime). In 2021, I was named an ACM Distinguished Member for "Outstanding Scientific Contributions to Computing."

23. My service record to the community includes serving as the co-chair for the USENIX Annual Technical Conference in 2020 (ATC'20); serving as the co-chair for USENIX File and Storage Technologies (FAST'15) in 2015 and on the FAST Conference Steering Committee from 2015 to 2023; serving on the ACM HotStorage Steering Committee since 2021; and serving as the co-chair in 2012 and on the Steering Committee of the ACM SYSTOR conference since 2012. I have served as an Associate Editor to the ACM Transactions on Storage ("TOS") journal from 2009 to 2022; in 2022, I was named the Editor-in-Chief for ACM's TOS journal.

24. I am a named inventor on four patents, two titled "Systems and Methods for Detection of New Malicious Executables" (U.S. Patent No. 7,487,544, issued February 3, 2009; and U.S. Patent No. 7,979,907, issued July 12, 2011); and two more titled "Multi-Tier Caching," (U.S. Patent No. 9,355,109, issued May 31, 2016; and U.S. Patent 9,959,279, issued May 1, 2018).

25. I have been disclosed as a testifying expert in 24 cases (including *inter partes* review ("IPR") proceedings) in the past four years. I have been deposed 14 times and testified in trial twice.

26. I have personal knowledge of the facts and opinions discussed in this declaration and believe them to be true. If called upon to do so, I would testify competently thereto. I have been warned that willful false statements and the like

are punishable by fine or imprisonment, or both. I have had no contact with any of the named inventors of the ’721 Patent.

27. My consulting company, Zadoks Consulting Services, is being compensated for my time at my current standard consulting rate. I am also being reimbursed for expenses that I may incur during the course of this work. My compensation is not contingent upon the results of my study and analysis, the substance of my opinions, or the outcome of any proceeding involving the Challenged Claim(s).

28. I have no financial interest in the Petitioner. I have no financial interest in the outcome of this matter or in any litigation involving the ’721 patent.

II. METHODOLOGY AND MATERIALS CONSIDERED

29. I have relied upon my education, knowledge and experience, as well as the other materials as discussed in this declaration in forming my opinions.

30. For this work, I have been asked to review U.S. Patent No. 10,574,721 (“’721 Patent”) (EX1001) including the specification and claims, and the ’721 Patent’s prosecution history (“’721 FH”) (EX1002). In developing my opinions relating to the ’721 Patent (EX1001), I have considered the materials cited or discussed herein, including those listed below:

Exhibit	Description
1001	U.S. Patent No. 10,574,721 to Bazzi *et al.* (“’721 Patent”)

Exhibit	Description
1002	File history of the '721 Patent ("'721 FH")
1004	U.S. Patent Application Publication No. 2012/0069401 to Fages *et al.* ("Fages")
1005	U.S. Patent No. 9,009,853 to Korycki *et al.* ("Korycki")
1006	International Publication No. WO 02/37333 to Jai ("Jai")
1007	U.S. Patent Application Publication No. 2014/0137184 to Russello *et al.* ("Russello")
1008	U.S. Patent No. 8,424,017 to O'Niell *et al.* ("O'Niell")
1009	U.S. Patent Application Publication No. 2003/0061257 to Cardona ("Cardona")
1010	U.S. Patent No. 7,493,414 to Tazuma ("Tazuma")
1011	*Curriculum vitae* of Erez Zadok, Ph.D.
1012	*Dictionary of Computer and Internet Terms*. 10th Edition. Barron's Educational Series, Inc. (2009). ("Barron's")
1013	*PC Basics Get a Great Start.* Gateway, Inc. (2002). ("Gateway")
1014	Dell, Inc. *Dell PowerEdge R620.* (2012). ("Dell")
1015	U.S. Patent No. 7,689,666 to Commons ("Commons")
1016	U.S. Patent No. 5,787,254 to Maddalozzo, Jr. *et al.* ("Maddalozzo, Jr.")
1017	Google Chrome Team. *20 Things I Learned About Browsers and the Web*. (2010). Available at: https://web.archive.org/web/20120127221853/https://dusk.geo.orst.edu/gis/Google_Book_20Things.pdf ("Google Team")
1018	U.S. Patent No. 8,914,813 to Sigurdsson *et al.* ("Sigurdsson")
1019	U.S. Patent No. 8,990,710 to Warner *et al.* ("Warner")
1020	US-CERT. *Banking Securely Online*. (2008). ("US-CERT")

Exhibit	Description
1021	Pope, M.B. *et al. The Domain Name System—Past, Present, and Future*. Communications of the Association for Information Systems: Vol. 30, Article 21. (2012). Available at: http://aisel.aisnet.org/cais/vol30/iss1/21 ("Pope")
1022	Microsoft Press. *Microsoft Exchange 2000 Server Resource Kit*. Pgs. 1-62. (2000). ("Microsoft Exchange")
1023	*Processes: A Brief Introduction*. (2006). Available at: https://www.linfo.org/process.html (retrieved August 13, 2025). ("Linux Information Project")
1024	Bradford Nichols, *et al. Pthreads Programming: A POSIXS Standard for Better Multiprocessing*. O'Reilly & Associates, Inc. (1996) ("O'Reilly")
1025	Devin Watson. *Linux Daemon Writing HOWTO*. (2004). Available at: https://homes.cs.aau.dk/~adavid/teaching/MTP-05/exercises/10/Linux-Daemon_Writting.pdf. Available at: https://web.archive.org/web/20120217144006/https://www.netzmafia.de/skripten/unix/linux-daemon-howto.html ("Watson")
1026	U.S. Patent No. 6,845,505 to Adunuthula *et al.* ("Adunuthula")
1027	U.S. Patent No. 8,230,399 to Vertes ("Vertes")
1028	U.S. Patent Application Publication No. 2014/0052683 to Kirkham *et al.* ("Kirkham")
1029	IBM Software Thought Leadership Whitepaper. *Native, web or hybrid mobile-app development*. (2012). Available at: https://public.dhe.ibm.com/software/sk/websphere/pdf/Native_Web_or_Hybrid_Mobile_Development.pdf ("IBM")
1030	Oracle Corporation. *DNS Basics*. Solaris Naming Administration Guide. (2010). Available at: https://docs.oracle.com/cd/E19455-01/806-1387/6jam692f2/index.html (retrieved August 20, 2025). ("Oracle – DNS")
1031	Martin Robillard. What Makes APIs Hard to Learn? Answers from Developers. IEEE Software. (November/December 2009). ("Robillard")

Exhibit	Description
1032	Larry L.P. and Bruce S.D. *In The Morgan Kaufmann Series in Networking, Computer Networks.* 5th Edition. (2012). Pp. 1-69. ("Computer Networks")
1033	Alok Sinha. *Client Server Computing Time Shared Computing.* Communications of the ACM. Vol. 35, No. 7. (1992). ("Sinha")
1034	Paul Clarke. *Introduction to Linux on IBM Power Systems.* IBM. (2021). Available at: https://developer.ibm.com/articles/intro-to-lop/ (retrieved August 18, 2025). ("Clarke")
1035	Stephen G.K. *Programing in C.* 3rd Edition. Sams Publishing. (2005). ("Kochan")
1036	The Bell System Technical Journal. Vol. 57, No. 6, Part 2. Pp. 1897-1094, 2807-2101. ("The Bell System Technical Journal")
1037	*Sockets Tutorial*. (2007). Available at: https://web.archive.org/web/20070120054454/https://www.linuxhowtos.org/C_C++/socket.htm (retrieved August 19, 2025). ("linuxhowtos")
1038	U.S. Patent Publication No. 2012/0317363 to Uola *et al.* ("Uola")
1039	Scott M.L. *Interprocess Communication in Unix and Windows NT.* Department of Computer Science, Brown University. (1997). Available at: https://cs.brown.edu/people/slewando/files/IPCWinNTUNIX.pdf ("Lewandowski")
1040	Oracle Corporation. *Appendix A Union Domain Sockets.* Network Interface Guide. (2010). Available at: https://docs.oracle.com/cd/E19455-01/806-1017/6jab5di3d/index.html#:~:text=UNIX%20domain%20sockets%20communicate%20only,properties%20visible%20to%20a%20user (retrieved August 20, 2025) ("Oracle – UNIX Domain Sockets")
1041	NoMachine. *NoMachine 4 Release Announcement*. (September 25, 2013). Available at: https://kb.nomachine.com/SU10K00101?s=nx+protocol (retrieved August 20, 2025). ("NoMachine")

Exhibit	Description
1042	NoMachine. *Using NX Client on a Linux desktop*. (February 5, 2012). Available at: https://web.archive.org/web/20120205022257/http://www.nomachine.com/screenshot/linux-client.php (retrieved August 20, 2025). ("NoMachine NX")
1043	NoMachine. *Article #AR10B00012*. (January 12, 2010). Available at: https://web.archive.org/web/20120114040432/http://www.nomachine.com/ar/view.php?ar_id=AR10B00012 (retrieved August 20, 2025). ("NoMachine AR").
1044	U.S. Patent Application Publication No. 2012/0023207 to Palchenko ("Palchenko")
1045	Richard Peterson. *Linux: The Complete Reference Fourth Edition*. The McGraw-Hill Companies. (2001). ("Peterson").
1046	U.S. Patent Application Publication No. 2007/0136579 to Levy *et al.* ("Levy")
1047	Conny Franke *et al.* "Isolated Web Browsing Enforcing OS-like Isolation for Browsers." December 16, 2005. Available at: https://www.researchgate.net/publication/277288272_Isolated_Web_Browsing_Enforcing_OS-like_Isolation_for_Browsers ("Franke")
1048	U.S. Patent Application Publication No. 2010/0218106 to Chen *et al.* ("Chen")
1049	Mohsen Zohrevandi *et al.* "Auto-FBI: A User-friendly Approach for Secure Access to Sensitive Content on the Web." ACSAC '13. December 9-13, 2013. New Orleans, LA. ("Auto-FBI")
1050	U.S. Patent No. 9,509,783 to Hayden ("Hayden")
1051	U.S. Patent No. 9,015,226 to Sullivan et al. ("Sullivan")
1052	U.S. Patent Application Publication No. 2005/0283767 to Blaku ("Blaku")

31. I have considered these materials from the viewpoint of a POSITA as of the priority date of the '721 Patent (EX1001). For the purposes of this declaration, I have been asked to assume that the priority date of the '721 Patent is December 6, 2013. I note that my opinions provided in this declaration are made from the perspective of a POSITA as of this priority date of the '721 Patent unless expressly stated otherwise. To the extent that I use any verb tense in this declaration that is present tense (*e.g.*, "a POSITA would understand" instead of "a POSITA would have understood"), such verb tense should be understood to be my opinion as of the '721 Patent's priority date (again, unless expressly stated otherwise). I merely use the present verb tense for ease of reading.

32. I have included the Proposed Grounds of Unpatentability from the Petition below:

Ground	**Claim(s) and Basis**	**References and Exhibits**
1	1-2 are obvious under AIA 35 U.S.C. §103	Fages (EX1004), Korycki (EX1005), Jai (EX1006), and Hayden (EX1050)
2	3-5 are obvious under AIA 35 U.S.C. §103	Fages (EX1004), Korycki (EX1005), Jai (EX1006), Hayden (EX1050), and Russello (EX1007)
3	6-8, 12-13, and 16-18 are obvious under AIA 35 U.S.C. §103	Fages (EX1004), Korycki (EX1005), Jai (EX1006), Hayden (EX1050), O'Niell (EX1008), and Cardona (EX1009)
4	9-10 and 14-15 are obvious under AIA 35 U.S.C. §103	Fages (EX1004), Korycki (EX1005), Jai (EX1006), Hayden (EX1050), O'Niell

		(EX1008), Cardona (EX1009), and Russello (EX1007)
5	11 is obvious under AIA 35 U.S.C. §103	Fages (EX1004), Korycki (EX1005), Jai (EX1006), Hayden (EX1050), O'Niell (EX1008), Cardona, and Tazuma (EX1010)

III. LEGAL FRAMEWORK

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

34. I have been informed that the Patent Trial and Appeal Board ("PTAB") now applies the Philips standard regardless of whether a patent has expired. I have been informed that under the Phillips standard, claim terms are to be given the meaning they would have to a person having ordinary skill in the art at the time of the invention, taking into consideration the patent, its file history, and, secondarily, any applicable extrinsic evidence (*e.g.*, dictionary definitions). I understand that, for purposes of this proceeding, Petitioner applies the ordinary and customary meaning of the claim terms as understood by a POSITA. I have also applied the plain meaning of the claim language as instructed by Phillips.

35. I have also been informed that a person cannot obtain a patent on an invention if the differences between the invention and the prior art are such that the

subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art. I have been informed that a conclusion of obviousness may be founded upon more than a single item of prior art. I have been further informed that obviousness is determined by evaluating the following factors: (1) the scope and content of the prior art, (2) the differences between the prior art and the claim at issue, (3) the level of ordinary skill in the pertinent art, and (4) secondary considerations of non-obviousness. In addition, the obviousness inquiry should not be done in hindsight. Instead, the obviousness inquiry should be done through the eyes of a person of ordinary skill in the relevant art at the time the patent was filed.

36. In considering whether certain prior art renders a particular patent claim obvious, counsel has informed me that I can consider the scope and content of the prior art, including the fact that one of skill in the art would regularly look to the disclosures in patents, trade publications, journal articles, industry standards, product literature and documentation, texts describing competitive technologies, requests for comment published by standard setting organizations, and materials from industry conferences, as examples. I have been informed that for a prior art reference to be proper for use in an obviousness analysis, the reference must be "analogous art" to the claimed invention. I have been informed that a reference is analogous art to the claimed invention if: (1) the reference is from the same field of

endeavor as the claimed invention (even if it addresses a different problem); or (2) the reference is reasonably pertinent to the problem faced by the inventor (even if it is not in the same field of endeavor as the claimed invention). In order for a reference to be "reasonably pertinent" to the problem, it must logically have commended itself to an inventor's attention in considering his problem. In determining whether a reference is reasonably pertinent, one should consider the problem faced by the inventor, as reflected either explicitly or implicitly, in the specification. I believe that all of the references that my opinions in this IPR are based upon are well within the range of references a person of ordinary skill in the art would consult to address the type of problems described in the Challenged Claims.

37. I have been informed that, in order to establish that a claimed invention was obvious based on a combination of prior art elements, a clear articulation of the reason(s) why a claimed invention would have been obvious must be provided. Specifically, I am informed that a combination of multiple items of prior art renders a patent claim obvious when there was an apparent reason for one of ordinary skill in the art, at the time of the invention, to combine the prior art, which can include, but is not limited to, any of the following rationales: (A) combining prior art methods according to known methods to yield predictable results; (B) substituting one known element for another to obtain predictable results; (C) using a known technique to improve a similar device in the same way; (D) applying a known technique to a

known device ready for improvement to yield predictable results; (E) trying a finite number of identified, predictable potential solutions, with a reasonable expectation of success; (F) identifying that known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to one of ordinary skill in the art; or (G) identifying an explicit teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine the prior art references to arrive at the claimed invention.

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

of these issues may be considered to determine whether there was an apparent reason to combine the known elements in the fashion claimed by the patent.

39. I also am informed that in conducting an obviousness analysis, a precise teaching directed to the specific subject matter of the Challenged Claim need not be sought out because it is appropriate to take account of the inferences and creative steps that a person of ordinary skill in the art would employ. The prior art considered can be directed to any need or problem known in the field of endeavor at the time of invention and can provide a reason for combining the elements of the prior art in the manner claimed. In other words, the prior art need not be directed towards solving the same specific problem as the problem addressed by the patent. Further, the individual prior art references themselves need not all be directed towards solving the same problem. I am informed that common sense is important and should be considered. Common sense teaches that familiar items may have obvious uses beyond their primary purposes.

40. I also am informed that the fact that a particular combination of prior art elements was "obvious to try" may indicate that the combination was obvious even if no one attempted the combination. If the combination was obvious to try (regardless of whether it was actually tried) or leads to anticipated success, then it is likely the result of ordinary skill and common sense rather than innovation. I am further informed that in many fields it may be that there is little discussion of obvious

techniques or combinations, and it often may be the case that market demand, rather than scientific literature or knowledge, will drive the design of an invention. I am informed that an invention that is a combination of prior art must do more than yield predictable results to be non-obvious.

41. I am informed that for a patent claim to be obvious, the claim must be obvious to a person of ordinary skill in the art at the time of the invention. I am informed that the factors to consider in determining the level of ordinary skill in the art include (1) the educational level and experience of people working in the field at the time the invention was made, (2) the types of problems faced in the art and the solutions found to those problems, and (3) the sophistication of the technology in the field.

42. I am informed that it is improper to combine references where the references teach away from their combination. I am informed that a reference may be said to teach away when a person of ordinary skill in the relevant art, upon reading the reference, would be discouraged from following the path set out in the reference, or would be led in a direction divergent from the path that was taken by the patent applicant. In general, a reference will teach away if it suggests that the line of development flowing from the reference's disclosure is unlikely to be productive of the result sought by the patentee. I am informed that a reference teaches away, for example, if (1) the combination would produce a seemingly inoperative device, or

(2) the references leave the impression that the product would not have the property sought by the patentee. I also am informed, however, that a reference does not teach away if it merely expresses a general preference for an alternative invention but does not criticize, discredit, or otherwise discourage investigation into the invention claimed.

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

44. I am further informed that secondary considerations evidence is only relevant if the offering party establishes a connection, or nexus, between the evidence and the claimed invention. The nexus cannot be based on prior art features.

The establishment of a nexus is a question of fact. While I understand that the Patent Owner here has not offered any secondary considerations at this time, I will supplement my opinions in the event that the Patent Owner raises secondary considerations during the course of this proceeding.

IV. OPINION

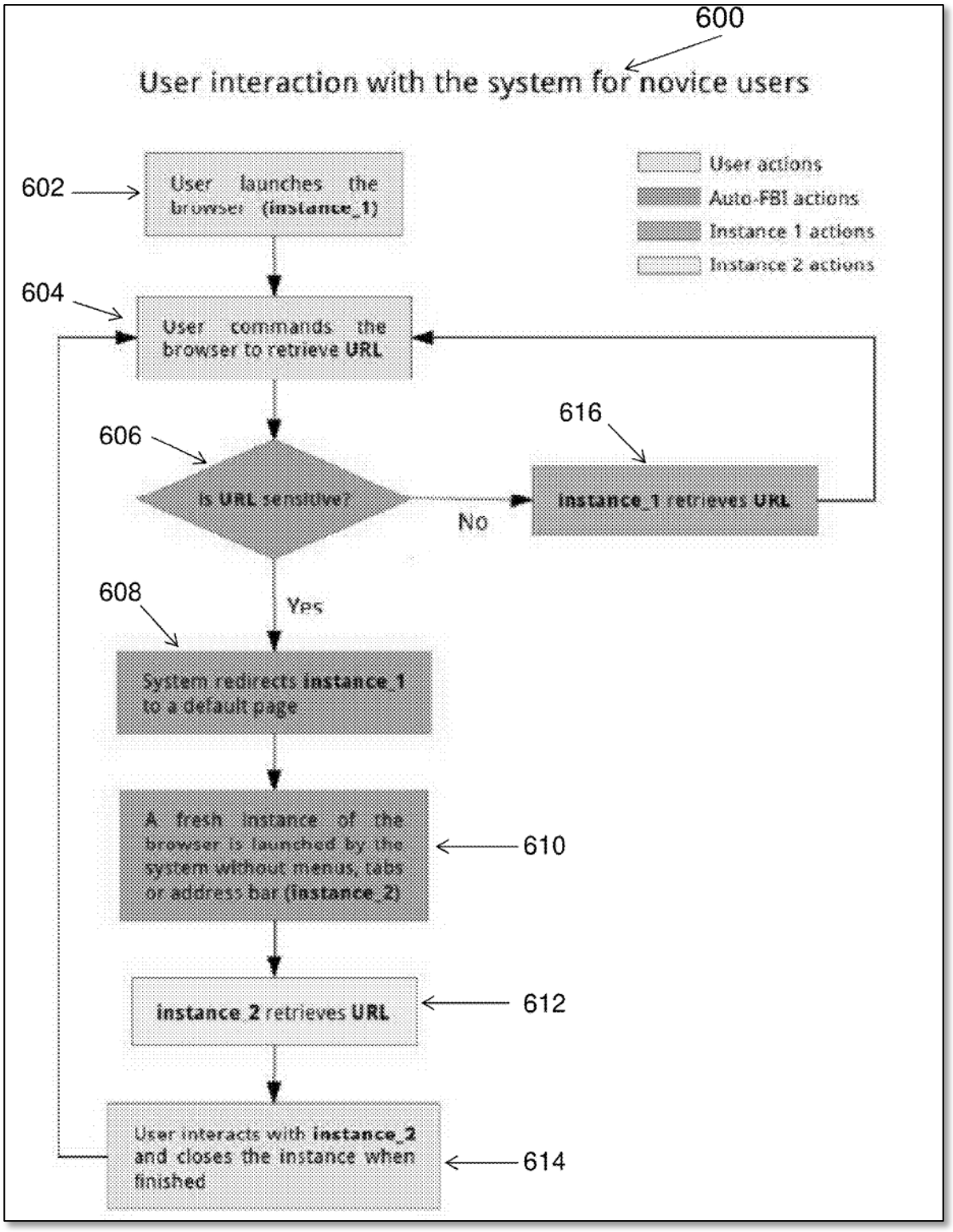
A. The '721 Patent

45. The '721 Patent generally describes systems and methods for compartmentalizing "different kinds of content with different browser instances." '721 Patent (EX1001), Abstract, 1:11-14. To "automate compartmentalization," the '721 Patent describes automatically launching a "fresh browser instance (FBI)" depending on whether the web content is sensitive or not. *Id*., Abstract ("For sensitive content, a Fresh Browser Instance (FBI) is automatically created to access the content."), 2:29-51 ("Aspects of the present disclosure involve an automatic fresh browser instance ("FBI") system ("FBI system") for implementing a process that automates compartmentalization practices for accessing different kinds of Internet content with different browser instances."), 3:24-32 ("The FBI system includes or otherwise implements a browser that processes content from a given origin to be a delegate of the origin. Web content is retrieved using web queries, which can be a simple query that specifies the address of the content in the form of a URL (uniform resource locator) which consists of a domain name and file path."),

4:32-53 (“The FBI system enables the processing of two types of sites for a novice user: sensitive sites and every other type of site. Access to a sensitive site is done through a dedicated FBI. Access to all other sites is done through separate browser instance(s) as it would be done without Auto-FBI.”).

46. In embodiments, when an initial web browser (*e.g.*, Chrome) sends a request to access a URL containing sensitive data, a browser extension (*e.g.*, JavaScript) associated with that web browser passes the URL to a daemon using a native application. *Id*., 2:29-51 (“the FBI system may use browser extensions to obtain the URL that the browser is attempting to access.”), 6:58-61 (“Enforcement in a browser extension: In a typical web browse [sic] (e.g., Chrome) extensions may be a combination of JavaScript and native code that is controlled by JavaScript.”), 10:8-37 (“All browser instances are equipped with an extension.”), Fig. 6, Claim 1.

47. The daemon “determines how to handle restricted URLs by either dispatching it to an already-running instance that is allowed to access the URL or creating a fresh new instance.” *Id*., 10:33-50, Fig. 6, Claim 1. This process is depicted in Figure 6 below:


600
User interaction with the system for novice users
User actions
Auto-FBI actions
Instance 1 actions
Instance 2 actions
602
User launches the browser (instance_1)
604
User commands the browser to retrieve URL
606
Is URL sensitive?
No
616
instance_1 retrieves URL
Yes
608
System redirects instance_1 to a default page
A fresh instance of the browser is launched by the system without menus, tabs or address bar (instance_2)
610
instance_2 retrieves URL
612
User interacts with instance_2 and closes the instance when finished
614

Id., Fig. 6.

48. In other embodiments, the ’721 Patent describes using DNS query resolution “to find out if [the user] is trying to connect to a restricted website.” ’721 Patent (EX1001), 9:41-56 (“To prevent the browser from accessing certain websites, the system keeps track of the DNS queries made by the browser to find out if it is trying to connect to a restricted website. The system can prevent the access by rewriting the IP addresses in the DNS response to point to a predefined address.”).

49. An experienced user sets up rules for handling certain content classes, the DNS query is compared against those rules, the “FBI system determines whether there is a different browser instance … associated with the class of URL[,]” and if so, opens a new instance to handle that URL. *Id.*, 6:26-45, 7:35–8:11 (“When a browser makes a DNS query, the tracer checks whether or not the requested network name is allowed to be accessed by the current browser instance.”), 9:58-61 (“To prevent the browser from connecting to a restricted website by using a hard-coded IP address, the system employs a white-list of IP addresses that the browser is allowed to connect to.”), Figs. 4 and 7, Claims 7, 11 (claim 11 recites DNS query).

B. Summary of the ’721 Patent’s Prosecution History

50. The ’721 Patent claims its earliest priority to provisional application 61/912,716, filed December 6, 2013.

51. To position the claims for allowance, Applicant amended the claims to recite, in part, a daemon, native application, and browser extension. ’721 FH

(EX1002), 363-368, 370-373 ("**Use of the daemon as recited and described is believed to be absent from the cited prior art references** … All of the independent claims now recite more clarifying features regarding how the daemon is implemented") (emphasis in original).

52. As explained in the Grounds in this Declaration (*see* §II above), Fages (EX1004), Korycki (EX1005), Jai (EX1006), Hayden (EX1050), Russello (EX1007), O'Niell (EX1008), Cardona (EX1009), and Tazuma (EX1010) disclose the allegedly patentable system and methods for compartmentalizing different kinds of web content with different browsing instances, utilizing claimed elements including a daemon, native application, browser extension, API call, and tracer.

C. Claim Construction

53. I have been informed by counsel that the first step in an unpatentability analysis involves construing the claims, as necessary, to determine their scope. Second, the construed claim language is then compared to the disclosures of the prior art. I am informed that claims are generally given their ordinary and customary meaning as understood by one of ordinary skill in the art at the time of the invention in light of the patent specification and prosecution history, *i.e.*, the intrinsic record. For purposes of my analysis, I have been asked to apply the plain and ordinary meaning as defined in this paragraph.

D. Level of a Person of Ordinary Skill in the Art

54. I understand that the ’721 Patent, the record of proceedings at the Patent Office (which I understand is called the “File History” or “Prosecution History”), and the teachings of the prior art, are evaluated from the perspective of a person of ordinary skill in the art (“POSITA”) at the time the claimed invention was made. I understand that the factors considered in determining the ordinary level of skill in the art include: (i) the levels of education and experience of persons working in the field; (ii) the types of problems encountered in the field; and (iii) the sophistication of the technology. I may also consider, if available, prior art solutions to the problems encountered in the art and the rapidity with which innovations are made in the relevant art.

55. I understand that a person of ordinary skill in the art is not a specific real individual, but rather a hypothetical individual having the qualities reflected by these factors. This hypothetical person has knowledge of all prior art in the relevant field as if it were arranged on a workshop wall and takes from each reference what it would teach to a person having the skills of a POSITA.

56. Based on my review and analysis of the ’721 Patent, the prior art cited herein, and the ordinary skill factors described in this section, a POSITA in the field of the ’721 Patent at the time of the earliest possible priority date (December 6, 2013) would have had a bachelor’s degree in computer science, electrical or computer

engineering, or an equivalent degree; and 2-3 years of work or research experience in computer security and networking. Additional education (*e.g.*, MS degree) might substitute for less experience and vice versa.

57. As of Dec. 6, 2013, I had more than ordinary skill in the art and meet at least these minimum qualifications to be a person having ordinary skill in the art as of the time of the claimed invention of the '721 Patent. I am, however, familiar with the skills and knowledge possessed by those I would have considered to be of ordinary skill in the art as of that date. *See* §I above, Educational Background and Qualifications.

58. My opinions provided in this declaration would not change in view of minor modifications to this level of ordinary skill.

V. BACKGROUND OF THE INVENTION AND PRIOR ART - LINUX

59. As I discuss below, a POSITA would have found it obvious and been motivated to modify Fages with other references with a reasonable expectation of success including because these combinations are compatible with Linux. *See, e.g.,* Motivation to Combine sections at Claims 1-5 §§VII.A.9 below, VII.A.10 below, VII.A.11 below, VII.A.12 below, VIII.A.2 below, VIII.A.4 below, VIII.A.5 below, VIII.A.6 below.

60. The '721 Patent refers to Linux. The inventors of the '721 Patent published a paper titled "Auto-FBI: A User-friendly Approach for Secure Access to

Sensitive Content on the Web." Auto-FBI (EX1049). The text of this article is fairly similar to the specification of the '721 Patent. In particular, the '721 Patent references ptrace, specifically referred to as "Linux ptrace API." '721 Patent (EX1001), 8:63-65, 7:15-20 ("To intercept system calls, the system includes a ptrace API available in Linux."). This API is well known in Unix/Linux operating systems for system call process tracing (pronounced "PEA-trace"). I have used ptrace, via tools like strace, well before even Linux existed; *see* §II above.

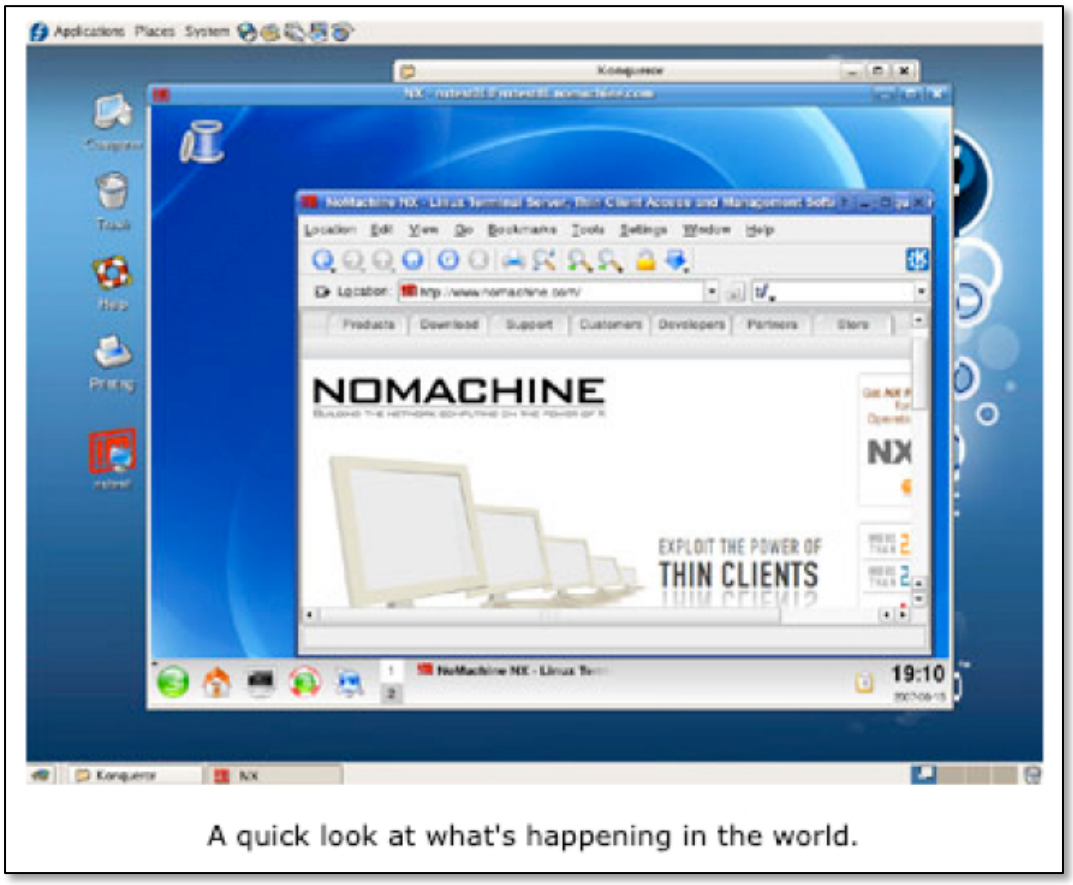
61. The paper on which the '721 Patent specification seems to be based on is even more expressly linked to Linux. Auto-FBI (EX1049), 2 ("We have developed a prototype system called Auto-FBI for 64-bit Linux. It works with Google Chrome and Mozilla Firefox."), 5 ("To intercept system calls, we employed the ptrace API available in Linux."), 5 ("Among hundreds of system calls available in Linux, we are interested in the following calls …"), 6 ("The implementation is entirely in user space and no modification to the Linux kernel is required."), 6 ("Using Linux ptrace API, we can trace every system call made by a child process to keep track of the application's access to system resources including network and file system."), 8 ("We have implemented the enforcer on 64-bit Linux system using the ptrace API.").

62. I note that prior art references used herein are also Linux-compatible. Linux itself is a variant of the Unix family of operating systems, which began even

earlier, in the early 1970s. Fages teaches "the taking of remote control over the Internet browsers executed on the server 20 is done by means of the NX protocol which is distributed in particular by the Italian firm NOMACHINE (www.nomachine.com)." Fages (EX1004), [0064]. It was known that NoMachine NX packages supported major operating systems, including multiple versions of the Linux Operating Systems. NoMachine AR (EX1043), ("Starting from NX 3.0.0, NoMachine, in order to cover the widest number of versions for the supported Linux Operating Systems, offers three families of RPM, DEB and compressed TAR packages."). NoMachine shared an example of "a Suse desktop session being accessed from a Fedora machine" to illustrate using NX client on a Linux desktop. NoMachine NX (EX1042), 1. The NX Client starts a session on the Fedora machine, as seen in the image below:

NoMachine NX (EX1042), 1.

63. Through the Fedora machine a user can access the Suse desktop, as seen in the figure below:



NoMachine NX (EX1042), 2.

64. In 2013, NoMachine "announced the official release of NoMachine version 4." NoMachine (EX1041), 1. NoMachine version 4 supports many Linux Operating System platforms, including common Linux distributions such as RedHat Enterprise, SLES, Open SUSE, Mandriva, Fedora Core, Debian GNU Linux, and Ubuntu. NoMachine (EX1041), 2. Using NoMachine's NX protocols, Fages is Linux-compatible.

65. Korycki is also compatible with Linux. Korycki teaches that "[t]he device 102 may be, for example, a mobile phone, a personal digital assistant

("PDA"), **a personal computer ("PC") (including**, for example, Windows™, Mac OS™ and **Linux™ PCs**), a laptop, a television, a gaming device or other embedded device able to connect to the network 112." Korycki (EX1005), 5:38-43.[1]

66. Similarly, Jai is also compatible with Linux. Jai teaches that its described processes are compatible with a broad variety of computing devices and operating systems. Thus, a POSITA would have understood that Jai is Linux-compatible, especially as Linux is a very popular operating system. I have personally used Linux extensively since the mid-1990s; *see* §I above and my CV (EX1011). Jai (EX1006), 13:3-7 ("The present invention may be used in a variety of computing devices. A 'computer device' is defined herein as any data processing device including microprocessors, conventional personal computers … The term is meant to be construed broadly.").

67. Indeed, Jai's URL-comparison functionality is commonly implemented in Linux-based systems. For example, Blaku describes similar URL-comparison functionality to Jai and implements that functionality on a Linux device. Blaku (EX1052), [0020] ("[E]very five seconds (or some other pre-selected or pre-defined

---

[1] All annotations and bold and/or bold-italics emphases in this Declaration are added unless stated otherwise.

period of time), the portion 60 will discern the identity of every homepage of every browser which is operatively installed within the computer 10. … The portion 60 will then (or concurrently with identifying the various browsers) determine whether a browser window is open. If a browser window is open (e.g., the browser or information associated with the browser is displayed on display 12), the portion 60 will determine if the 'target URL' portion of that browser is the same as that browser's previously identified homepage."), *id.* ("Then the portion 60 continually determines which browsers have been open (if any) and for each of these open browsers, the portion 60 continually monitors the respective homepages of these browsers and each of the respective target URLs which are associated with these browsers and respectively compares the target URLs with the homepages."), [0066] ("Such applications could include software on a Linux® type system, on an Apple® type system, on a Windows® type system, or on a personal data type device.").

68. Sullivan also has similar functionality to Jai and likewise describes URL comparison implemented on a system running in a Linux operating environment. Sullivan (EX1051), 3:35-38 ("In one embodiment, the method further comprises rendering, by the server computer system, content for the URL based on whether the window that originated the first request is a new window or an existing window."), 5:41-48 ("In certain embodiments, web browser 106 can enable users to access multiple web resources simultaneously via multiple browser windows 108."),

11:16-18 (“Server computer system 810 can run any consumer or server operating system such as Microsoft Windows, Apple OS X, Linux, Solaris, and the like.”).

69. Additionally, Hayden is compatible with Linux. Hayden (EX1050), 38:32-55 (“Any process descriptions, elements, or blocks in the flow diagrams described herein and/or depicted in the attached FIGURES should be understood as potentially representing modules, segments, or portions of code which include one or more executable instructions for implementing specific logical functions or steps in the process.”), *id.* (“Accordingly, general purpose computing devices may be configured to implement the processes, algorithms, and methodology of the present disclosure with the processing and/or execution of the various data and/or components described above.”), 15:44-16:7 (“Similar schema may also be developed for any number of device operating system or software framework processing operations.”).

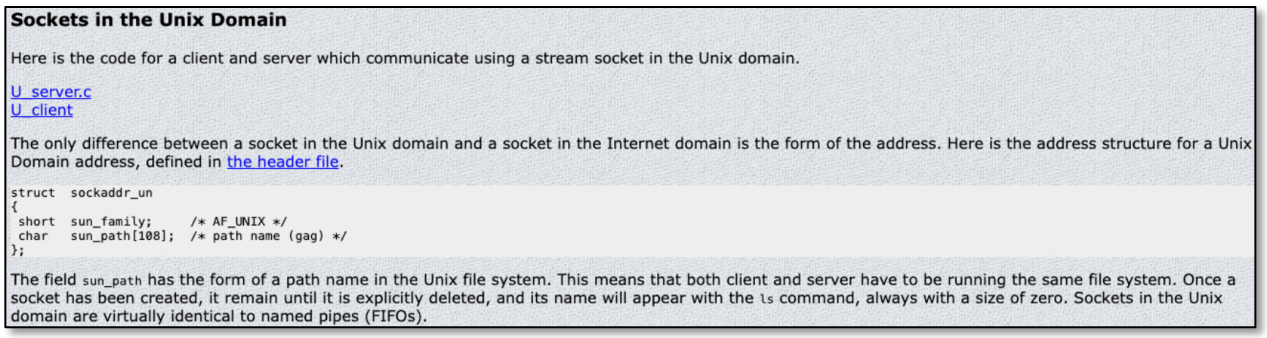
70. Indeed, Hayden’s browser check functionality is commonly implemented in Linux-based systems due to their popularity and widespread use. For example, Levy has similar functionality to Hayden and describes implementing similar browser checks in a Linux operating environment. Levy teaches when a user enters a URL, a manifest is checked to determine if a webpage may be retrieved “under the existing browser instance” or whether a new browser instance needs to be opened. Levy (EX1046), [0015], [0022], [0051], [0054], [0055], [0059]-[0067],

[0126], [130]. Levy teaches each browser instances in its own virtual machine. Levy (EX1046), [0046], [0062], [0070]. Further, Levy teaches each virtual machine domain executes a guest operating system, such as Linux. Levy (EX1046), [0059]-[0062] ("Each browser instance executes in its own Xen virtual machine. In FIG. 4, the first two browser instances on VMs 112 and 120 are running versions of the Konqueror™ Web browser, which was ported to the new browser architecture, on top of a Linux guest OS."). Levy also teaches performance testing using Linux. Levy (EX1046), [0110]-[0124] ("Linux version 2.6.10 was booted either directly on the CPU or in Xen virtual machines."), *id.* ("Konqueror was used on Linux for this test.").

71. Russello teaches "a security system and method for a Linux-based operating system[.]" Russello (EX1007), [0015], [0018] ("… an operating system running on a device comprising a Linux-based kernel and a system architecture defined by a Linux layer associated with the kernel …"). Russello is Linux-compatible, as Russello specifically teaches application to a Linux-based operating system.

72. O'Niell teaches creating Unix Domain Sockets and Linux Domain Sockets, which are Linux-compatible inter-process communication (IPC) mechanisms. O'Niell (EX1008), 2:51-56 ("The disclosed method extends the use of sockets, such as Unix domain Sockets, to determine if a particular process is

running. In one embodiment, the disclosed method initializes an application that launches a particular process. The method then creates a Unix domain socket for the particular process."), 5:47-54 ("Network stack 240 includes a domain socket handler 245 that creates a Socket, such as a Unix domain Socket or Linux domain Socket, in response to a request from application 230 or daemon process 235."); Linuxhowtos (EX1037), 1 ("This is a simple tutorial on using sockets for interprocess communication."). Linuxhowtos (EX1037) teaches a tutorial on "Sockets in the Unix Domain":



Sockets in the Unix Domain

Here is the code for a client and server which communicate using a stream socket in the Unix domain.

U_server.c
U_client

The only difference between a socket in the Unix domain and a socket in the Internet domain is the form of the address. Here is the address structure for a Unix Domain address, defined in the header file.

```
struct  sockaddr_un
{
 short  sun_family;     /* AF_UNIX */
 char   sun_path[108];  /* path name (gag) */
};
```

The field `sun_path` has the form of a path name in the Unix file system. This means that both client and server have to be running the same file system. Once a socket has been created, it remain until it is explicitly deleted, and its name will appear with the `ls` command, always with a size of zero. Sockets in the Unix domain are virtually identical to named pipes (FIFOs).

Linuxhowtos (EX1037), 5.

73. Unix Domain Sockets were compatible with the Linux Operating System, as "Linux is quite reliable and highly compatible with UNIX". Barron's (EX1012), 9.

74. Lastly, Cardona teaches a multithreaded universal daemon (MUD) that can handle multiple threads and tasks concurrently. Cardona (EX1008), [0036] ("The exemplary daemon according to the invention is multithreading, that is, it spawns multiple threads or tasks that exist concurrently."), [0053] ("The resulting

daemon is referred to as a multithreaded universal daemon (MUD), because it is capable of simultaneously monitoring many connections, up to the number that was specified as a parameter in the create function."). It was known that daemons were written for the Linux Operating System environment. Watson (EX1025), 1 ("This document shows how to write a daemon in Linux using GCC"). A POSITA would have known that Cardona's MUD is Linux-compatible.

VI. BACKGROUND OF THE TECHNOLOGY

75. The features described and claimed in the '721 Patent were routine as of December 6, 2013. In this section, I briefly summarize the technology background for a few key features relevant to my opinions in this Declaration.

A. Computers and Servers

76. It was well known before the priority date of the '721 Patent that a computer is a machine consisting of two major parts, "MEMORY and the central processing unit (CPU)." Barron's (EX1012), 6. The inclusion of a processor in a computer is fundamental for the computer to execute instructions. Barron's (EX1012), 6.

77. The CPU is the core component in a computer, as it "interprets and carries out instructions, performs all computations, and controls the devices connected to the PC." Gateway (EX1013), 11. In the early 2000s, for example, there were a variety of CPUs available on the market, including Intel's Pentium and

Celeron processors, and AMD's Athlon and Duron processors. Gateway (EX1013), 11 ("There are a variety of CPUs available on the market. A well-known company is Intel®, which produces the Pentium® and Celeron® processors. Another company is AMD®, which produces the Athlon and Duron processors.").

78. "Most computers use a VON NEUMANN ARCHITECTURE, which means that programs and data are stored in the same kind of memory." Barron's (EX1012), 8. This includes a server, which is "a computer that provides services to another computer[.]" Barron's (EX1012), 12.

79. The Von Neumann architecture is also referred to as a "classic computer architecture." Palchenko (EX1044), [0003]. Palchenko teaches that a "conventional PC is separated into two parts according to the Von Neumann's architecture: Module 1, for example, and Terminal 9. The module 1 represents a central processor 10, a memory 11, I/O controllers of drivers 12, which are all combined in a system (see FIG. 3)." *Id.*, [0034]. Palchenko's Fig. 3 below depicts processor 10 and memory 11 as part of computer 1:

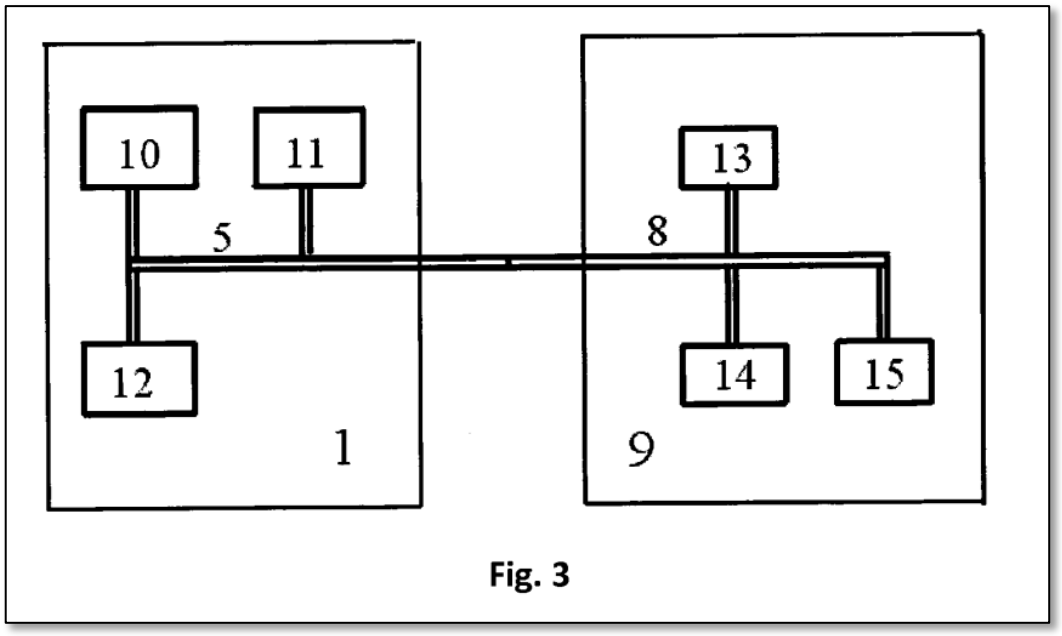


Fig. 3

Id., Fig. 3.

80. It was well known before the priority date of the '721 Patent that a server is a computer. Barron's (EX1012), 12 ("**server** a computer that provides services to another computer (called the *client*).") (emphasis original). As a computer, a server fundamentally includes a processor to execute instructions. *Id.*, 6 ("Fundamentally, a computer is a machine that can store instructions and execute them."), ("The CPU spends its time retrieving instructions from memory and doing whatever those instructions say."). Server computers consisting at least of processors and memory were commercially available. One example was the Dell PowerEdge R620, which was commercially available at least as early as 2012. Dell (EX1014), 1-2 ("October 2012"). "With its hyper-dense memory (up to 768GB in a 1U form factor) and powered by Intel® Xeon® E5-2600 processors, the R620 really

shines when handling virtualization, high performance computing (HPC), and workgroup collaboration applications." *Id.*

B. Web Browsers, Browser Extensions, and Browser Plugins

81. Browsers and browser extensions were well known before the priority date of the '721 Patent. Browsers were known to allow a user to view files located on other computers through the internet. As taught by Commons, "[a] file … is accessed by requesting the address of the file, the Uniform Resource Locator (URL)." Commons (EX1015), 1:26-31. In particular, Commons teaches that "[t]ransmission Control Protocol and Internet Protocol (TCP/IP) allow computers on the Internet to communicate … The URL consists of a protocol, a domain name associated with an IP address, and a file name … A Domain name consists of a number of elements designed to correspond to an IP address." *Id.*, 1:46-57.

82. It was known that the Hypertext Transfer Protocol (HTTP) was commonly used for browsers to make web connections, as taught in Maddalozzo, Jr. (EX1016). "Client and server communicate using the functionality provided by Hypertext Transfer Protocol (HTTP) … Active within client is a first process browser 72 which establishes the connections with server … which presents information to the client in the form of HTTP responses." *Id.*, 6:64- 7:15.

83. Browser extensions were known to provide extended functionality to browser programs. Notably, browser extensions add features to the browser that are

important to and meet the needs of the user. Google Team (EX1017), 20 ("Browser extensions let you add new features to your browser – literally *extending* your browser. This means that you can customize your browser with the features that are most important to you. Think of extensions as ways of adding new superpowers to what the browser can already do. These superpowers can be mighty or modest, depending on your needs.") (emphasis original).

84. This enabled rapid evolution of web technologies: rather than modifying native source code for a browser, compiling it, disseminating new executables, and restarting them, one could merely download ready-made small extensions directly at runtime into a running web browser. Users could try out many extensions, keeping the ones they like, and discarding the rest, thus also keeping resource consumption such as memory to a bare minimum. Browser extensions could also be dynamically updated when new versions were released, all with minimal to no disruption to users.

85. Sigurdsson teaches extensions being used for communication between browser programs and other programs on the device. "[A] system enables cross-browser extensions. The system includes a translation module configured to receive a cross-browser command included in a browser extension." Sigurdsson (EX1018), 2:59-62.

86. Browser extensions were also known to communicate with other computer processes through application programming interfaces (APIs). Sigurdsson teaches that its "system also includes a native browser configured to receive the commands formatted according to a native browser extension application program interface to execute the browser extension." *Id.*, 2:67-3:3.

87. Browser plugins were known before the '721 Patent as a method of extending browser functionality utilizing executable code. Plugins were known to be able to perform similar behavior to browser extensions. For example, Warner teaches "a browser extension may include [a] plug-in or add-on installed at the user's device so as to extend the functionality of a web browser also installed and executable at the device." Warner (EX1019), 6:35-39.

C. Sensitive Web Content and Web Browser Security

88. When using a browser to view web content, there are online attacks such as malware, phishing, and other online attacks to stay aware of and be protected from. Google Team (EX1017), 26 ("… browser security helps protect you from malware, phishing, and other online attacks[.]").

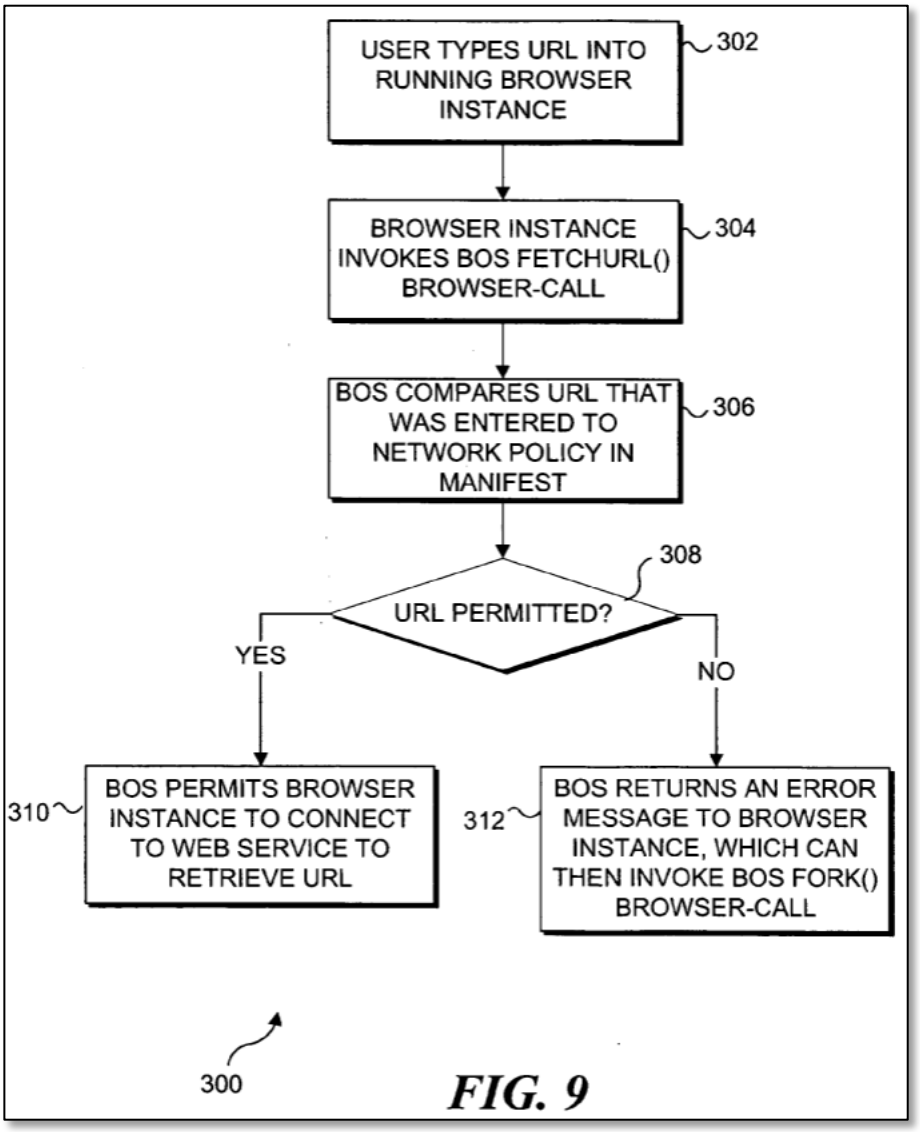
89. When it comes to online banking, sensitive information such as a bank account number, credit card numbers, names, and passwords can be the target of attacks. Google Team (EX1017), 29 ("When you browse the web and perform transactions online, two security risks to be aware of are malware and phishing."),

("The email contains phishing links that look like they go to your bank's website, but really take you to an imposter website. There you're asked to log in, and inadvertently reveal your bank account number, credit card numbers, passwords, or other sensitive information to the bad guys."), ("Malware … you may visits a page that installs software on your computer without even asking. The software is really designed to steal credit card numbers or passwords from your computer[.]"); US-CERT (EX1020), 1 ("Several types of electronic fraud specifically target online banking. Some of the more popular types are described below: ***Phishing attacks***") (emphasis original), 2 ("***Malware***") (emphasis original), ("***Pharming***") (emphasis original).

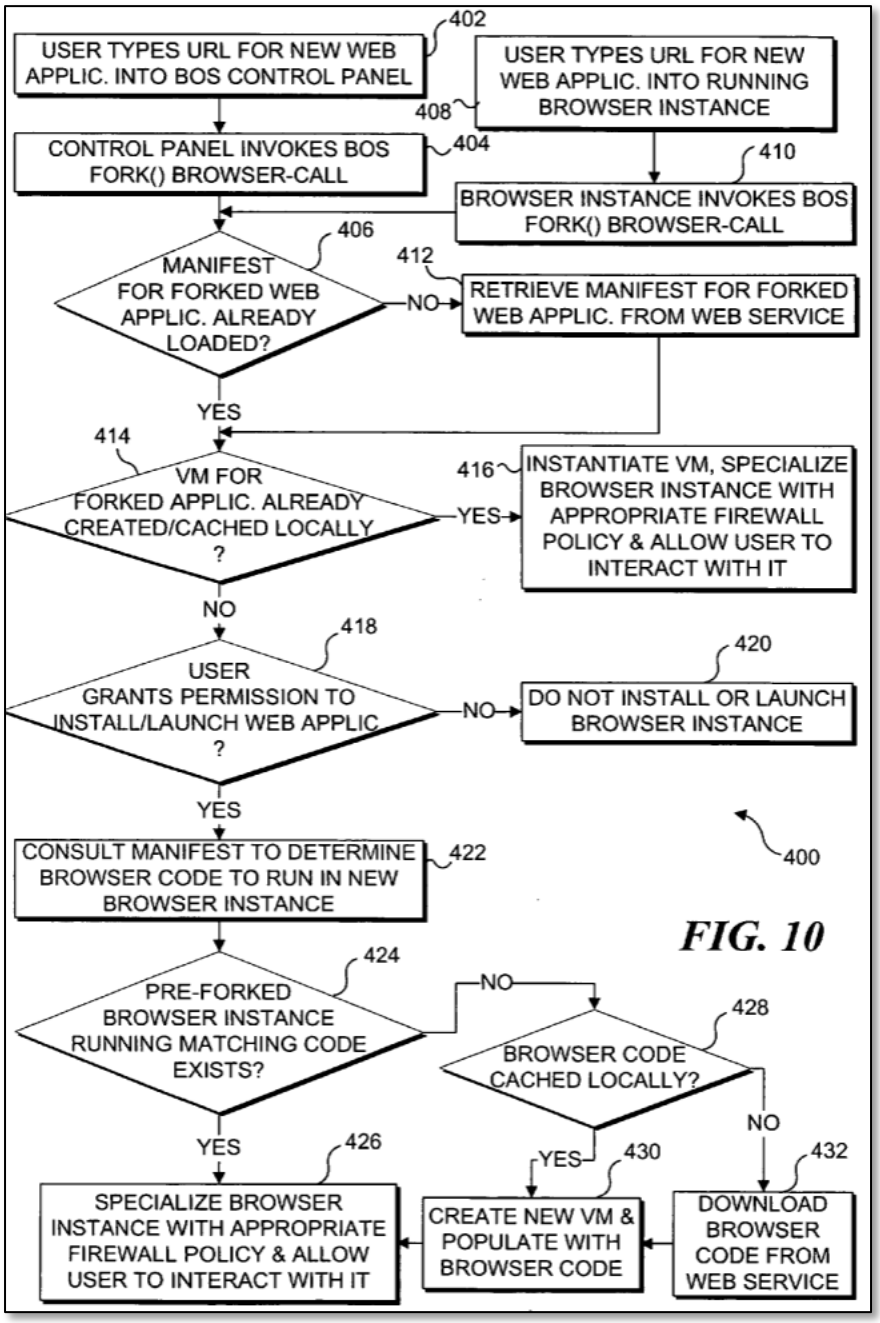
90. I also note that web browser security/confinement was known before the priority date of the '721 Patent. For example, Levy teaches how a user enters a URL and checks a manifest to determine if the webpage may be retrieved "under the existing browser instance" or whether a new browser instance must be initiated. Levy (EX1046), [0126] ("FIGS. 9 and 10 respectively illustrate flowcharts 300 and 400 that include exemplary logical steps for fetching a Webpage with a Web service with an existing browser instance running under the BOS, and for forking a new Web application and browser instance."), [0051] (teaching a plurality of isolated browser instances and conventional browsers), ("The Figure illustrates three web applications, including an online bank application, a Web mail application, and an

online radio application, and their corresponding isolated browser instances 60, 64, and 68, respectively. Two of the browser instances 60 and 64 contain conventional browsers (Mozilla™ 62 and Konqueror™ 66), while the third is executing a custom radio client software application 70, instead of a conventional Web browser."), [0054] ("To access the Web, a browser instance invokes a BOS system call that fetches Web documents over HTTP. The BOS will service the connection only if the document falls within the network policy … If not, the BOS refuses the request. If the document is allowed by the manifest of a different Web application, the BOS gives the user the option of loading it into that Web application's browser instance."), [0055] (teaching sandboxing for web browsers), ("Sandboxes provide private virtual disks to browser instances, and the BOS maintains the state of these disks between invocations of the Web application."), [0059]-[0067] (also teaching sandboxing and utilizing a Linux environment), [0130] ("The new browser architecture implementation discussed above provides a new Web browsing system intended to improve safety and security for Web users. … each Web application is isolated within its own virtual machine sandbox, removing the need to trust Web browsers and the services they access. Virtual machine sandboxes contain the damage that can be caused by malicious or vulnerable browsers."), [0015], [0022].

91. The logical steps of retrieving a webpage under an existing browser instance or in a new browser instance, as taught by Levy, is showed in Figures 9 and 10 below:



Levy (EX1046), Fig. 9.



Levy (EX1046), Fig. 10.

92. In another example of web browser security/confinement, Franke teaches separating "distinct applications into separate OS processes, and appropriately restrict[ing] access to disk-based state." Franke (EX1047), 1. Franke

also teaches that "a malicious page should not have access to the state, either in memory or on disk, of the user's browser-based banking application."). *Id.* Franke teaches "a clever attacker could craft a page that harnesses browser flaws to inspect the Document Object Model (DOM) of another currently displayed page and examine, for example, banking information." Franke (EX1047), 2, 3 ("Browsing Domain A grouping of related browsing activity. For example, clicking within a news website constitutes one browsing domain, while shopping at Amazon.com would constitute another."), *id.* ("We want to classify browser network activity into distinct domains according to a given policy, so that we can enforce separation based on the process policy."), *id.* (We desire that each distinct web application, either in a tab or separate browser window, be protected from the other running applications.").

93. Franke also teaches system call interception. *Id.*, 4 ("We chose to use a system call interception scheme.), *id.* ("This technique is sufficiently low-level that we can intercept any browser-initiated communication we'd like."). Franke teaches utilizing Linux and using a kernel module called Mozwatcher for intercepting system calls for browser instances. *Id.*, 4-6.

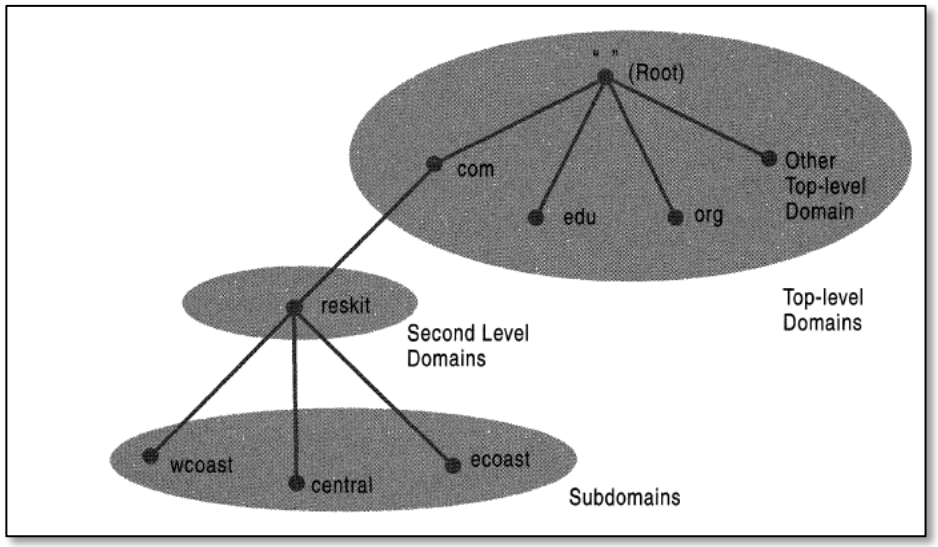
D. Domain Name System (DNS)

94. The DNS was created in the 1980's, well before the priority date of the '721 Patent and is "a hierarchical namespace" that forms "a critical component of

the Internet's infrastructure … that turns an Internet host name (domain name) into the unique series of numbers which constitute an Internet Protocol (IP) address for each specific domain name." Microsoft Exchange (EX1022), 84; Pope (EX1021), 3.

95. When searching in a browser, "the DNS translates a URL or web address (like 'www.google.com') into the right IP address to contact (like '74.125.19.147')" to display the desired webpage. Google Team (EX1017), 37-38; Pope (EX1021), 3 (describing how the DNS "is the translation system" between domain names and IP addresses).

96. The DNS namespace hierarchy is illustrated below:



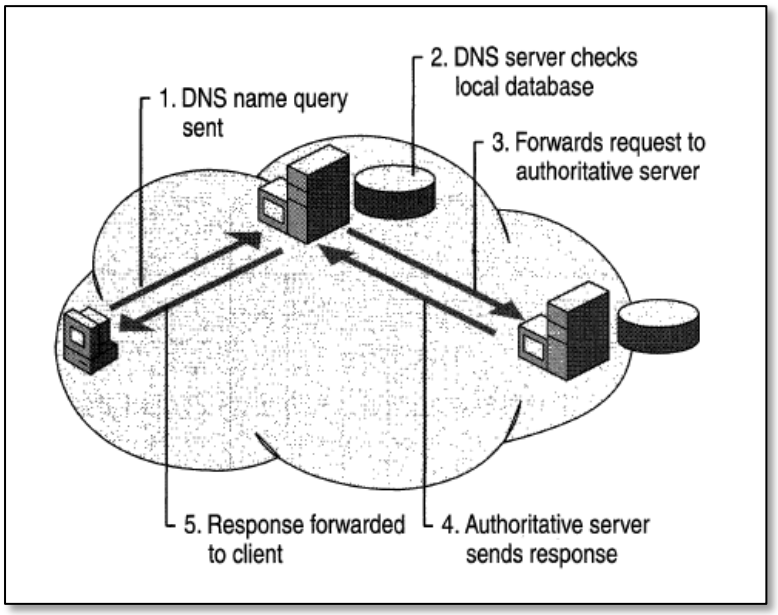
Microsoft Exchange (EX1022), Fig. 1.11, 85.

97. Here, top-level domain (TLD) name examples include "COM" meaning "Commercial organization" or "GOV" meaning "Government institution". *Id.*, Table 1.26.

98. "The unique name of the host, representing its position in the hierarchy, is called its Fully Qualified Domain Name (FQDN)" like "ftpsrv.wcoast.reskit.com" for example. *Id.*, 84. "One common way to resolve a host name to an IP address is to use a locally stored database file that contains IP-address-to-host-name mappings." *Id.*, 86.

99. DNS servers store the mappings between IP addresses and FQDN's. "To enable the querying of a DNS server by a host computer, a component called the DNS resolver is enabled and configured with the IP address of the DNS server. The DNS resolver is a built-in component of TCP/IP protocol stacks supplied with most network operating systems[.]" *Id.*, 87. "To obtain the IP address of a server that is authoritative for the FQDN, DNS servers on the Internet go through an iterative process of querying multiple DNS servers until the authoritative server is found." *Id.*, 88.

100. The steps for resolving a DNS query are illustrated in the figure below:



Id., Fig. 1.12, 88.

101. Specifically, the DNS resolver of a DNS client, such as the DNS software of a user's computer, sends a DNS name query with the FQDN (1) to the DNS server, which compares the DNS query's FQDN against its stored addresses (2). *Id.*, 88; Oracle – DNS (EX1030), 1. The IP address is returned to the client if found or forwarded by the DNS server to another server "authoritative for the FQDN" if not (3). The process repeats until an authoritative DNS server returns the "resolved IP address" to the original DNS server (4) to forward to the client (5). *Id.*, 88.

E. Daemons

102. Daemons were well known before the priority date of the '721 Patent. A daemon is a program or process "that runs continuously in the background, or is activated by a particular event," and not under direct control of a user. Barron's

(EX1012), 124; Linux Information Project (EX1023), 2 ("Daemons are a class of processes that run continuously in the background, rather than under the direct control of a user."). "Daemons are usually launched automatically while a computer is booting up and then wait in the background until their services are required." Linux Information Project (EX1023), 2. Daemons "typically respond to hardware activity, to network requests or to other programs by performing specified tasks." *Id.*, 2.

103. "A common design model for a UNIX server daemon is to accept requests and fork a child image of itself to process the request." O'Reilly (EX1024), 8. A daemon designed to use multiple processes would benefit from threading when "the benefits of concurrency outweigh the overhead of using separate processes in the application," as "threading is bound to improve its performance because threads involve less overhead." *Id.*

104. A daemon in the Linux environment can be written in C. Watson (EX1025), 1 ("This document shows how to write a daemon in Linux using GCC."), 2 ("Throughout this HOWTO, a very simple daemon will be built in C."). Although the code requires proper order of execution specific to a daemon, it does not specify utilizing additional hardware. *Id*., 2 ("As we go along, more code will be added, showing the proper order of execution required to get a daemon up and running"). Watson teaches that "GCC 3.2.2 or higher" and "Linux Development headers and

libraries" are packages needed "on your Linux machine to develop daemons," and does not specify utilizing additional hardware. *Id*.

105. As taught in Watson's tutorial, a program has specific steps to first perform and indicate that it is a daemon. *Id.*, 2-3 ("When a daemon starts up, it has to do some low-level housework to get itself ready for its job. This involves a few steps: Fork off the parent process[;] Change file mode mask (umask)[;] Open logs for writing[;] Create a unique Session ID (SID)[;] Change the current working directory to a safe place[;] Close standard file descriptors[;] Enter actual daemon code[.]"). After the specific initial steps, the program "basically told Linux that [it's] a daemon." *Id.*, 6. This differentiates a daemon from a program that does not run in the background.

106. It was known to utilize a daemon to regulate and filter web communications. Methods of monitoring data and communication between the browser and the kernel were useful and known at the time, including daemons. "[T]he HTTP daemon recognizes the request … and sends the reply to the client." Adunuthula (EX1026), 6:40-43.

107. In another example, Chen teaches automatically utilizing a "preferred" browser based on various preferences and that a daemon will automatically determine the most suitable browser to handle a URL request from a user. Chen (EX1048), [0008] ("… selecting a preferred browser from a plurality of browsers

according to the at least one preference tag …"), [0053] ("When the client 110 request and receives the web page from web server 140, the daemon program 1102 may intercept and determine the suitable browser for the received web page.").

108. Daemons were ubiquitous in Unix/Linux operating systems, so much so that a naming convention emerged, to append a 'd' to their program/process name (*e.g.*, smtp**d** for the email daemon, http**d** for the web server daemon, nfs**d** for the network file system daemon, and many more). I personally managed, developed, and maintained such daemons starting in the 1990s. *See* §I above.

F. System Trace

109. Process tracing is a well-known technique in Unix/Linux operating systems, via a special system call called "ptrace" (pronounced "PEA-trace"). I have used ptrace, via tools like strace, well before even Linux existed; *see* §I above.

110. Most operating systems, including Linux, provided a ptrace system call that could trace system calls. Vertes (EX1027), 4:27-34 (describing how Linux and "most operating systems today, including embedded operating systems, provide a ptrace system call, and the tracing of system calls with ptrace, in their kernel.") The ptrace call "provides a user-space controller visibility and control on another (controlled) process." *Id.*, 4:39-42. "[T]he request PTRACE_TRACEME executed by the ptrace system call tells the kernel that the calling process wants to be traced."

Id., 4:55-58. "The PTRACE_SYSCALL request … makes the process to be traced to stop after the next system call." *Id.*, 4:63-65.

G. Native Applications

111. Native applications were well known before the priority date of the '721 Patent. Kirkham teaches that "[w]eb applications generally run as part of a web browser that is separately loaded onto a computing device, whereas native applications are executable apart from a web browser. Web applications may be easy to code and to use, while native application may provide a richer experience that is not constrained by limits of HTML coding or the particular browser that may be loaded on a device." Kirkham (EX1028), [0003]; IBM (EX1029), 2 ("Upon initialization, the native app interfaces directly with the mobile operating system, without any intermediary or container. The native app is free to access all of the APIs that are made available by the OS vendor and, in many cases, has unique features and functions that are typical of that specific mobile OS.").

H. Application Programming Interface (API)

112. APIs were in use before the '721 Patent. "An API is the interface to implemented functionality that developers can access to perform various tasks. APIs support code reuse, provide high-level abstractions that facilitate programming tasks, and help unify the programming experience (for example, by providing a uniform way to interact with list structures)." Robillard (EX1031), 27. APIs give

the "syntax" for services defined by different protocols to work on a given computer system. Computer Networks (EX1032), 38. Sinha teaches that "carefully designed" APIs "remain unchanged across various platforms (i.e., vendor- independent) within a specific desktop environment (e.g., PC LAN vs. Unix workstation LAN). Porting of applications, based on these APIs, from one platform to another platform [thus] becomes an easy task …." Sinha (EX1033), 8; IBM (EX1029), 2.

113. Computer Networks teaches that "the *socket interface* originally provided by the Berkeley distribution of Unix" is a network API "supported in virtually all popular operating systems" beyond its native operating system. It is also foundational for "language-specific interfaces, such as the Java socket library." Computer Networks (EX1032), 38.

I. Linux Operating System

114. The operating system Linux was well known before the priority date of the '721 Patent. Barron's (EX1012), 2 ("© Copyright 2009"), 10 ("Linux"). "Linux is quite reliable and compatible with UNIX; as a result, it is very popular with universities, Internet service providers, and small businesses that need multi-user computing at minimum cost." *Id.* Linux "can be given away for free provided they are complete and intact, but most users prefer to purchase commercially produced CD-ROMs containing Linux together with application software." *Id*.

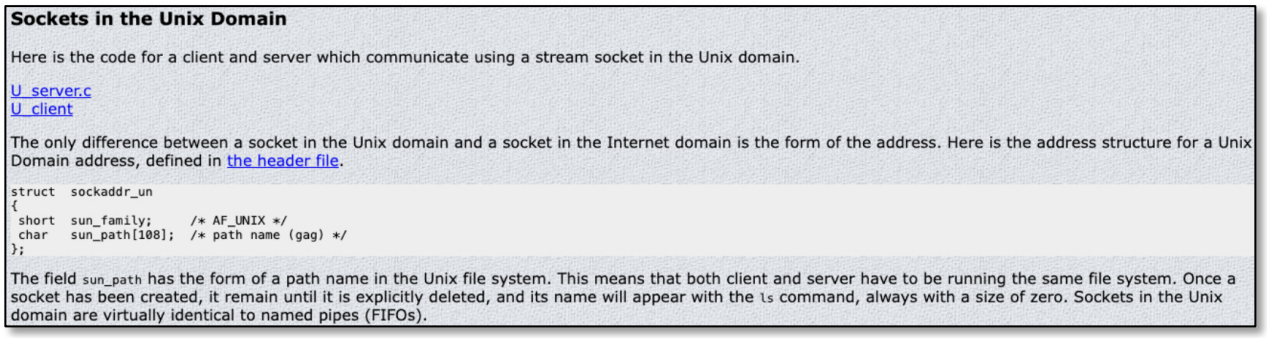
115. Linux is "a freely distributed UNIX-compatible operating system for PCs and a number of other processors." *Id.*, 10. For example, the PowerPC processor supports running the Linux Operating System. Clarke (EX1034), 1 ("The IBM RISC processors have continued to mature and compete ever since, with early IBM Power® processors, IBM PowerPC® processors, today's IBM Power9 processors, and the recently announced tenth-generation Power10 processors. These processors have and continue to … support several world-class enterprise-level operating systems including IBM AIX®, IBM i, and Linux[.]"). "From 1994 to 2006, PowerPC CPUs were used in the Apple Macintosh." Barron's (EX1012), 11.

J. C Programming Language

116. Developed in the early 1970s and standardized in the 1990s, the C Programming Language was well known before the priority date of the '721 Patent. Kochan (EX1035), 22 ("The C programming language was pioneered by Dennis Ritchie at AT&T Bell Laboratories in the early 1970s" and "in 1990, the first official ANSI standard definition of C was published" which the International Standard Organization "adopted" since "C is used around the world[.]"). C is a "general-purpose structured programming language" that "enable[s] the user to 'get in close' with the hardware and deal with the computer on a much lower level." *Id*., 22-23. As a "general-purpose structured programming language," C "provides the user with an enormous amount of power and flexibility." *Id.*, 23. The UNIX operating system,

"developed at Bell Laboratories, had C as its "standard" programming language[,]" and "well over 90% of the operating system itself was written in the C language." *Id.*, 22; The Bell System Technical Journal (EX1036), 2088 ("The UNIX system is written in the programming language, C, in which most user programs and subsystems are written.").

117. It was known that sockets for interprocess communication could be programmed in the C programming language. Linuxhowtos (EX1037), 1 ("You are here: Programming->C/C++"), ("This is a simple tutorial on using sockets for interprocess communication."). Linuxhowtos taught a tutorial on programming sockets in C. *Id*., ("You are here: Programming->C/C++"), ("This is a simple tutorial on using sockets for interprocess communication."). "When a socket is created, the program has to specify the *address domain* and the *socket type*." *Id*., (emphasis original). The Unix domain is a widely used address domain where two processes that share a common file system communicate. *Id*., ("There are two widely used address domains, the *unix domain*, in which two processes which share a common file system communicate, and …") (emphasis original). Linuxhowtos taught an example for a client and server to communicate using a stream socket in the Unix domain, with C programming files:



Sockets in the Unix Domain

Here is the code for a client and server which communicate using a stream socket in the Unix domain.

U_server.c
U_client

The only difference between a socket in the Unix domain and a socket in the Internet domain is the form of the address. Here is the address structure for a Unix Domain address, defined in the header file.

```
struct  sockaddr_un
{
 short  sun_family;     /* AF_UNIX */
 char   sun_path[108];  /* path name (gag) */
};
```

The field `sun_path` has the form of a path name in the Unix file system. This means that both client and server have to be running the same file system. Once a socket has been created, it remain until it is explicitly deleted, and its name will appear with the `ls` command, always with a size of zero. Sockets in the Unix domain are virtually identical to named pipes (FIFOs).

Id., 5.

VII. OPINION: GROUND 1

A. Obviousness over Fages, Korycki, Jai, and Hayden

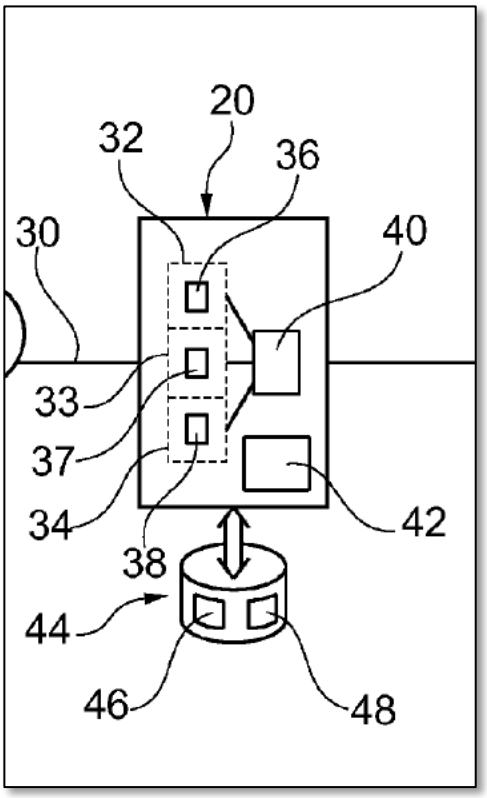
118. Claims 1-2 would have been obvious over Fages, Korycki, Jai, and Hayden.

1. Overview of Fages

119. U.S. Patent Application Publication No. 2012/0069401 to Fages *et al.* ("Fages") was published March 22, 2012 (EX1004, cover).

120. Fages teaches "parallel execution of several Internet browsers in execution environments that are independent of one another" such that if a "URL belongs to a pre-recorded list of URLs of sensitive Internet sites to which connection from the first execution environment is not authorized," Fages launches a new browser in parallel in a "second execution environment" "to connect to the sensitive Internet site identified by this URL[.]" Fages (EX1004), [0044]-[0046]; *see also id.*, [0098]-[0099], [0104], [0109]-[0110].

121. Fages teaches a system for providing browser security and a server computer for implementing its system. Fages (EX1004), [0050] ("FIG. 1 is a schematic illustration of a system for browsing on the Internet."), [0058]-[0063], [0084]-[0089], claims 24-25. As depicted in Figure 1 below, Fages's system 2 comprises server 20 including server module 40.



Id., Fig. 1 (excerpt); *see also id.*, [0055]-[0056].

122. Fages's server 20 includes an "information-recording medium comprising instructions for executing" instructions that are "executed by an electronic computer" including software programs and instructions stored in memory (*e.g.*, memory 44). *Id.*, [0044], [0064]-[0065], [0096]. Notably, Fages

teaches that "each user station has an electronic computer 50 executing a client module 52 of the software program for taking remote control. For example, the computer 50 is that of a **central processing unit** of a desktop computer." *Id.*, [0065]-[0069].

123. Fages teaches that an administrator loads server 20 with data ("*receive data*") defining non-secure content class ("*first web content class*") and data defining secure content class ("*second web content class*") such as sensitive/secure websites for online banking. *Id.*, [0084]-[0086]; *see also id.*, [0018]-[0024], [0044]-[0046], [0072]-[0074]. In this way, the administrator loads server 20 with data defining first and second web content classes that each define a plurality of URLs ("*first plurality of Uniform Resource Locators (URLs)*," "*second plurality of URLs different than the first plurality of URLs*") where each content class defines a different set of URLs. *Id.*, [0084]-[0086]; *see also id.*, [0018]-[0024], [0044]-[0046], [0056].

124. Fages describes the same understanding of "browser instance"/"browser instance type" as the '721 Patent. Fages teaches the generating of a first browser session/instance ("*first browser instance*") in an execution environment. *Id.*, [0003] ("an access server connected to the Internet and capable of the parallel execution of **several instances of an Internet browser**"), [0018]-[0020], [0044]-[0048].

125. Fages teaches that Internet browser instance 36 in execution environment 32 accesses content from the plurality of URLs of non-sensitive content and "a stage [90] of unsecured browsing is carried out on the Internet network 4." *Id.*, [0088]-[0091]. When a user submits a request for a new URL from the first browser instance while browsing in non-secure session S_4, "this new URL is compared with the URLs contained in all the lists L_{i1}." *Id.*, [0091]. Fages teaches that if the requested URL is on the list of sensitive/secure URLs, then the access to this sensitive/secure URL is blocked for Internet browser 36 of execution environment 32 of session S_4 because an "unsecured session enables connection with any Internet site except the sensitive Internet sites." *Id.*, [0088]-[0089], [0091]-[0092], [0072]-[0074]. Another browser instance may be utilized to access a sensitive URL, as detailed in Claim 1(e). *See* §VII.A.14 below, Claim 1(e); *see also* Fages (EX1004), [0098]-[0099], [0109]-[0110]. Otherwise, if the URL is non-sensitive (*i.e.*, not contained in a sensitive URL list), "then the connection to this Internet site is authorized" and the URL is accessed, for example, using browser 36 in execution environment 32 and session S_4. Fages (EX1004), [0090]-[0093].

2. Overview of Korycki

126. U.S. Patent No. 9,009,853 to Korycki *et al.* ("Korycki") was filed March 7, 2012 (EX1005, cover).

127. Korycki teaches implementing a computer system that performs Active Control List (ACL) security to allow or disallow a user to access certain web content. Korycki (EX1005), 8:41-9:8, 9:34-10:22. Korycki teaches utilizing a pub-sub.js "widget" 206 ("browser extension"), a browser plugin 208 implemented using native technology ("native application"), and a daemon 218 ("daemon") that performs background processing including security functions related to the ACL security. *Id.*, 7:21-33 ("The publish-subscribe daemon application 218 is a native part of the service provider client application and is always running in the background at the device 102."), 9:60-10:22, 6:40-7:6.

128. Korycki teaches performing computer security by checking whether a user is allowed to access certain content, such as a web application, which is "executed in a first web browser instance." *Id.*, 1:10-21, 1:66-2:31. Specifically, Korycki teaches a "**daemon**" 218 that accesses a list of approved partners for a service web application and determines "whether the domain of the partner associated with the partner web application 212 (which was indicated in the subscriber request message) is present in the ACL [Access Control List], i.e., whether the service provider has approved the partner web application 212 for communication with the service provider web application 204." *Id.*, 8:41-9:8, 9:60-10:22.

129. Korycki teaches that "only desired partner web applications are allowed to communicate with the service provider web application," and "any other web application (other than those on the list of partners approved by the service provider) may be prevented from participating in communication with the service provider web application." *Id.*, 1:66-2:31. In one example, Korycki teaches "comparing a domain associated with the partner web application with the entries of the list of partners approved by the service provider" to determine "whether the partner web application is approved for communication." *Id.*, 3:9-13.

130. Korycki teaches that daemon 218 is utilized for performing background processes related to the device security. *Id.*, 7:21-33 ("The publish-subscribe daemon application 218 is a native part of the service provider client application and is always running in the background at the device 102."), 9:60-10:22, 6:66-7:6. A POSITA would have understood that a daemon such as daemon 218 is a background process/program that performs its functions automatically without requiring user intervention. *See* §VI.E above. For example, Korycki's daemon 218 runs as a background process on a computer system. Korycki (EX1005), 7:21-33. Thus, a POSITA would have understood that Korycki's daemon 218 can execute the tasks it was programmed to perform without requiring a user to specifically request that the functions be performed. Likewise, other daemons would have performed the tasks they are programmed to perform.

131. Korycki's annotated Fig. 2 below depicts daemon 218 communicating with browser plugin 208 ("native application") and pub-sub.js widget 206 ("browser extension"), which are both depicted as part of web browser instance 202 (referred to as Browser 1 below). As indicated in Fig. 2, daemon 218 and browser plugin 208 are technology "native to the device platform of the device 102 (*e.g.*, C++)." *Id.*, 7:21-33.

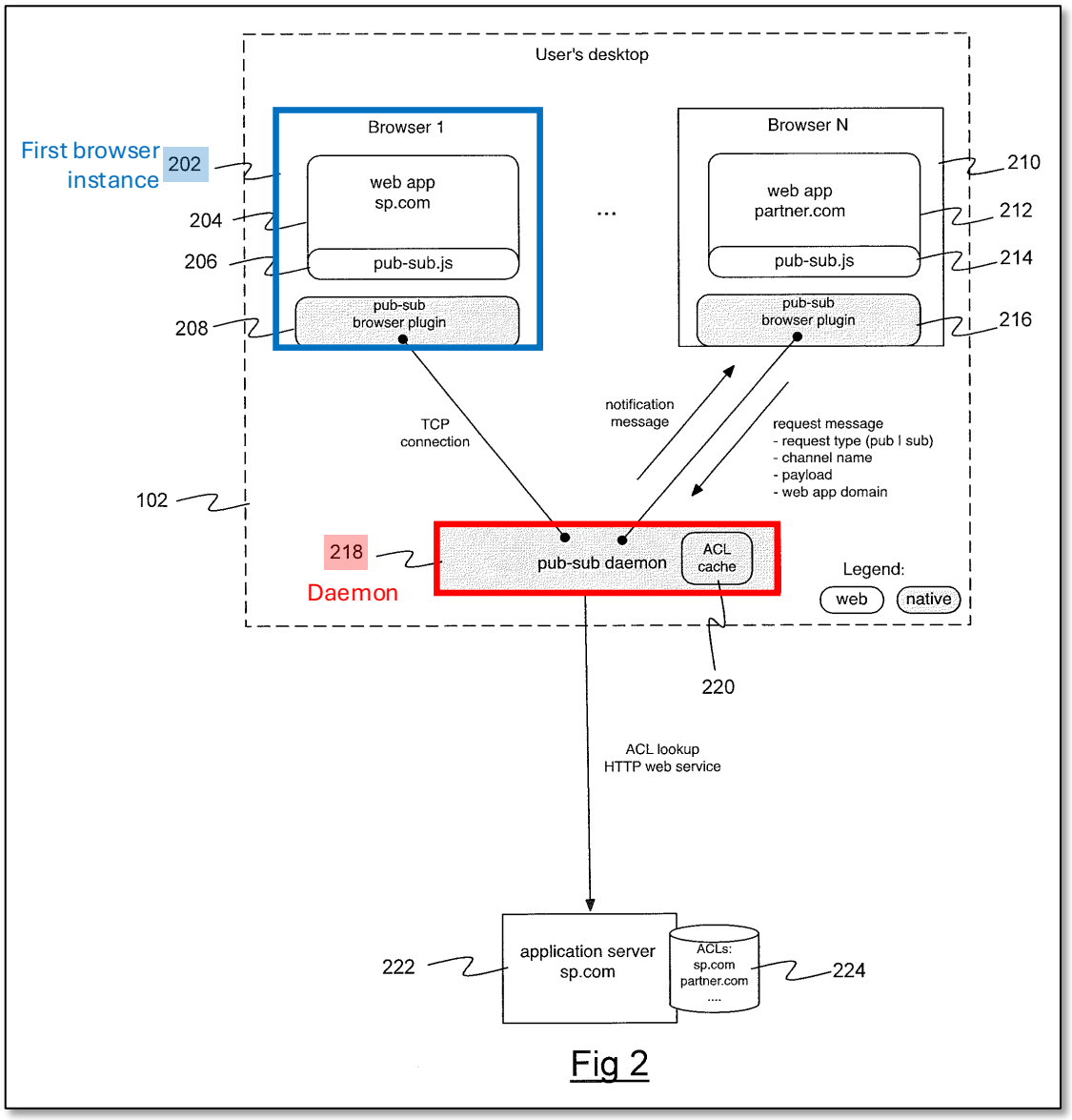


Fig 2

Id., Fig. 2 (annotated).

132. Korycki teaches that daemon 218 is outside browser 1, for example, and that "signals can be transmitted between the publish-subscribe daemon application 218 and the publish-subscribe browser plugin 208 and between the publish-subscribe daemon application 218 and the publish-subscribe browser plugin 216 using the publish-subscribe signaling scheme" which "may be achieved by establishing respective Transport Control Protocol (TCP) connections." *Id.*, 7:34-49, Fig. 2.

3. Overview of Jai

133. International Publication No. WO 02/37333 to Jai ("Jai") was published on May 10, 2002.

134. Jai teaches a computer system that performs a URL comparison to detect if a requested URL is already open in a browser instance. Jai (EX1006), 3:22-4:2, 4:20-21 ("Figure 8 is a schematic diagram of a web browser illustrating a web site having a minimized application of a second website"), 12:8-13:7 ("Another aspect of the present invention is illustrated in Figure 8 and prevents redundant applications of the same website 20 from being opened at a given time. When a user inputs a URL and requests that the associated site b[e] opened, a search is made of the other loaded websites 20. If a loaded site having the same URL is already open, it will be displayed to the user without a second application being opened."), 14:3-5

("Many of the examples and illustrations describe and show a single embodiment. However, two or more applications of the invention may be used in a single device."), Fig. 8. Jai teaches that in this way "duplicate applications of the same site may also be avoided." Jai (EX1006), 3:22, Abstract. This is because "prior to accessing and loading another site, the present invention searches other open applications and determines whether the new request is actually a duplicate of an already open application. If it is a duplicate, the existing application is displayed and the second requested application of the same site is deleted." Jai (EX1006), 3:22-4:2.

135. Specifically, Jai's Fig. 8 "is a schematic diagram of a web browser illustrating a web site having a minimized application of a second website." Jai, 4:20-21. Jai's Fig. 8 annotated below depicts address box 36 (for typing in a URL), website 20 (website a user is currently browsing), location 83 (a "currently loaded website," specifically www.ericsson.com, which is the same website 20 the user is currently browsing), and location 81 (another "currently loaded website" that is "loaded and has been minimized," specifically www.cellularphones.com):

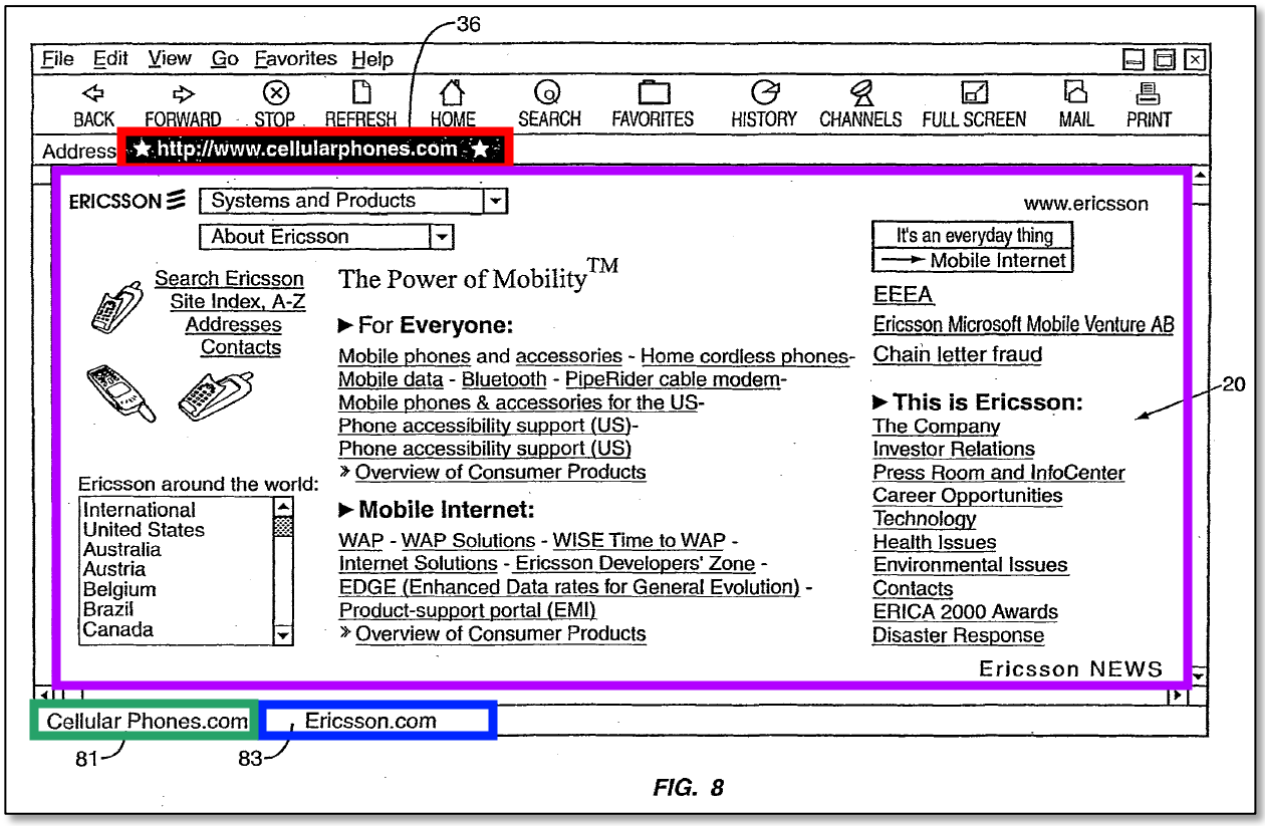


FIG. 8

Jai (EX1006), Fig. 8 (annotated), 12:14-18.

4. Overview of Hayden

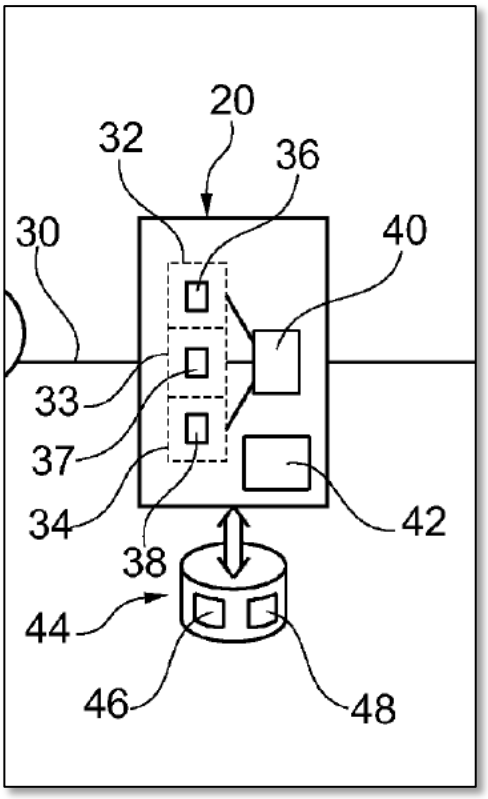
136. U.S. Patent No. 9,509,783 to Hayden was filed on January 26, 2012.

137. Hayden teaches determining if a particular browser instance is already launched, and if a particular browser instance is not already launched, Hayden teaches launching the particular browser instance. Hayden (EX1050), 35:4-36:32, Fig. 13. Specifically, Hayden describes that "FIG. 13 illustrates an example routine 1300 implemented by a virtual machine manager 182 or other component of the network computing provider 107 for routing requests to browser instances that have been customized to handle the request." Hayden (EX1050), 35:4-8. Furthermore,

"if a user of a client computing device 102 submits a request for http://www.xyzwebsite.com, the instance selection module 196 of the virtual machine manager 182 can determine which of the available network computing components to route the request to, based on whether a customized browser virtual machine image 186 has been created for the requested site and, if so, whether a browser has been instantiated from the image 186." Hayden (EX1050), 35:8-15, Fig. 13.

5. Claim 1(Pre): "A system, comprising: a processor, operable to:"

138. To the extent the preamble is limiting, Fages teaches a system for providing browser security and a server computer for implementing its system. Fages (EX1004), [0050] ("FIG. 1 is a schematic illustration of a system for browsing on the Internet."), [0058]-[0063], [0084]-[0089], claims 24-25. As depicted in Figure 1 below, Fages's system 2 comprises server 20 including server module 40.



Id., Fig. 1 (excerpt); *see also id.*, [0055]-[0056].

139. Fages's server 20 includes an "information-recording medium comprising instructions for executing" instructions that are "executed by an electronic computer" including software programs and instructions stored in memory (*e.g.*, memory 44). *Id.*, [0044], [0064]-[0065], [0096].

140. A POSITA would have understood or found obvious that server 20 would have included a processor (*e.g.*, CPU) to execute those instructions such as those to implement server module 40 and anti-virus software program 42, and instructions stored in memory 44. Including a processor in a computer was well-known, and processors were standard components in computers. *See* §VI.A above.

A server computer, like a desktop computer, would have included a processor to execute programming instructions. *See* §VI.A above. A POSITA would have understood or found obvious that a server computer would have had a processor because a processor is a standard component of the von Neumann computer architecture, which is the standard design for modern computers. *See* §VI.A above. A POSITA would have understood from Fages's description of server 20 that the server has the typical von Neumann computer architecture because the server stores instructions to be executed to perform its functions.

141. Notably, Fages teaches that "each user station has an electronic computer 50 executing a client module 52 of the software program for taking remote control. For example, the computer 50 is that of a **central processing unit** of a desktop computer." Fages (EX1004), [0065]-[0069], *see also id.* [0010] disclosing that an execution environment "designates a state of the access server defined by the **computer resources** allocated solely to the execution of a specific computer program as well as a set of environmental values and variables to which this program has access. **The computer resources are for example the processor time**, the random-access and virtual memory spaces and a data storage space on the hard disk drive. Thus, the execution environments partition the computer resources in such a way that a data program executed in one execution environment cannot act on the resources allocated to another execution environment."). Fages teaches that server

20 is "capable of the parallel execution of several Internet browsers in execution environments that are independent of one another." Fages (EX1004), [0045].

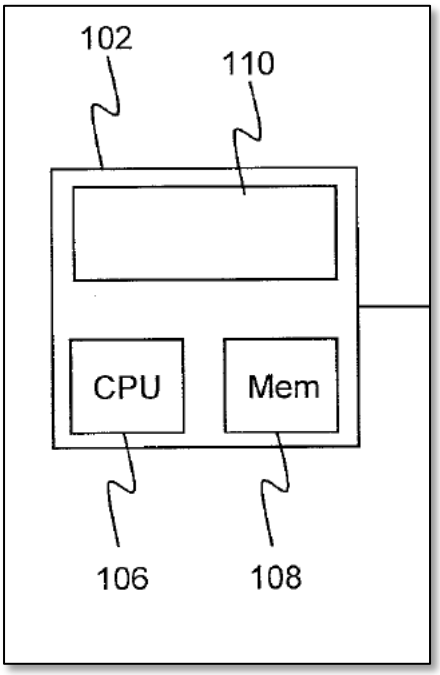
142. Thus, a POSITA would have understood or found obvious that Fages's server would have had a processor for executing the execution environments and because the execution environments would have had processor time allocated as a computer resource, and a processor would have been utilized to execute computer programming instructions during said allocated processor time. Moreover, while not expressly discussed, a POSITA would have understood that Fages's server would have had a processor for executing programming instructions similarly to how the user stations include a processor for executing programming instructions.

143. A POSITA would have understood that a user computer (*i.e.*, general purpose computer) would have been based on the von Neumann computer architecture and thus would have included a processor to execute programming instructions. *See* §VI.A above. Similarly, a POSITA would have understood that, while not expressly described in Fages, a server computer would have included a processor to execute its programming instructions stored in memory. *See* §VI.A above. It was well known to a POSITA that server computers included a processor for executing instructions, as this is a basic function of a server (or any) computer. *See* §VI.A above. Further, Fages also teaches server 20 executes program

instructions stored in memory 44, which a POSITA would have understood or found obvious to have been performed by a processor in the server computer.

a. Korycki Teaches a Processor

144. Additionally or alternatively, Korycki teaches "a processor (CPU) 106" in a user station computer (device 102). Korycki (EX1005), 5:32-38, 13:36-38 ("The components 202 to 218 shown in FIG. 2 are preferably implemented in software for execution by the CPU 106 on the device 102"). A POSITA would have understood that Korycki's processor performs similar security functions as Fages's server 20. *Id.*, 3:20-41, 5:46-48, 13:7-34. Korycki's processor 106 is depicted below:



Id., Fig. 1 (excerpt).

145. Thus, a POSITA would have understood or found obvious that server 20 would have natively included a processor to perform its functions in the same way that Korycki's computer includes a processor to perform its functions. That is, a POSITA would have understood that Fages, which stores programming instructions in memory, would have utilized the same mechanism (*i.e.*, processor) to execute its programming instructions as Korycki, which teaches a similar computer system and also stores its programming instructions in memory. A POSITA would have been motivated to modify Fages's server computer to include a processor to execute its programming instructions, consistent with Korycki's teachings of using a processor to execute programming instructions. Next I discuss a motivation to combine Fages with Korycki to utilize a processor to execute programming instructions.

146. **In sum**[2], Fages alone or in view of Korycki teaches "A system [Fages's computer system including server 20], comprising: a processor [Fages's native server processor or Fages's server modified to include a processor such as Korycki's], operable to:".

---

[2] Each summary paragraph is exemplary and overviews an example of how the prior art teaches and/or renders obvious the claims. My analysis is not limited to these examples and encompasses the other examples described herein.

b. Motivation to Combine Fages with Korycki (Including a Processor in a Server Computer)

147. A POSITA would have been motivated to modify Fages with Korycki's teachings to utilize a processor in a server computer to execute programming instructions. Modifying Fages in this way would have been a straightforward and well-understood modification to result in utilizing a processor to execute programming instructions stored in memory. This modification would have combined prior art elements according to known methods to yield the predictable result of a server computer performing Fages's functions by executing instructions via a processor. For example, prior art elements such as Fages's server computer and memory would have been combined with a processor, such as the processor taught by Korycki or another suitable processor for a server computer, which would have been a commercially available, off-the-shelf component. Further, this use of a known technique would have improved the similar computing devices in the same way. Including a processor would have allowed Fages's server to execute programming instructions and perform its server computer functions.

148. This modification would have had a reasonable expectation of success because processors were well known, off-the-shelf components that were ubiquitous in computers before the priority date of the '721 Patent. *See* §VI.A above. A POSITA would have chosen a suitable processor for Fages's server 20 from among available processors at the time (or would have chosen a suitable available server

already including a processor), thus achieving a reasonable expectation of success. A POSITA would not have had to personally fabricate a server computer or personally physically add a processor to Fages's server, although such would have been achievable by a POSITA. Indeed, as I discussed in §I above, I had extensive experience with computer hardware such as processors before the priority date of the '721 Patent. *See* §I above, Educational Background and Qualifications. However, as I opined, a POSITA would not have had to personally modify a computer to include a processor, as server computers were commercially available with processors already included. *See* §VI.A above.

6. Claim 1(a): "receive data defining a first web content class and a second web content class, the first web content class defining a first plurality of Uniform Resource Locators (URLs), and the second web content class defining a second plurality of URLs different than the first plurality of URLs, the second web content class further being associated with sensitive information presenting a security risk;"

a. "receive data defining a first web content class and a second web content class"

149. Fages teaches an administrator loads server 20 with data ("*receive data*") defining non-secure content class ("*first web content class*") and data defining secure content class ("*second web content class*") such as sensitive/secure websites for online banking. Fages (EX1004), [0084]-[0086] ("Initially, at a step 80, an administrator of the system 2 gets connected to the server 20 by means of the station 24 and the network 22. The administrator uses this connection to define the table 70

as well as the different lists L_{i1} L_{i2} and C_i. After having defined the different possible Internet sessions through the server 20, the administrator records the sessions in the files 48."); *see also id.*, [0018]-[0024], [0044]-[0046], [0072]-[0074].

150. In this way, the administrator loads server 20 with data defining first and second web content classes that each define a plurality of URLs ("*first plurality of Uniform Resource Locators (URLs)*," "*second plurality of URLs different than the first plurality of URLs*") where each content class defines a different set of URLs. *Id.*, [0084]-[0086]; *see also id.*, [0018]-[0024], [0044]-[0046], [0056] ("The server 8 is a sensitive Internet server. The term 'sensitive' designates, in this embodiment, a server that must be protected against second type attacks. For example, the Internet server 8 is an Internet server that enables online handling of bank accounts.").

151. As I opine in the next section, Fages teaches an unsecure/non-sensitive session S_4 where a user may access the set of all URLs not included in the secure/sensitive URL lists L_{i1}. The secure-sensitive URL lists L_{i1} are accessible only via a respective secure session S_1-S_3, where secure/sensitive URL list L_{11} corresponds to secure session S_1, secure/sensitive URL list L_{21} corresponds to secure session S_2, and secure/sensitive URL list L_{31} corresponds to secure session S_3.

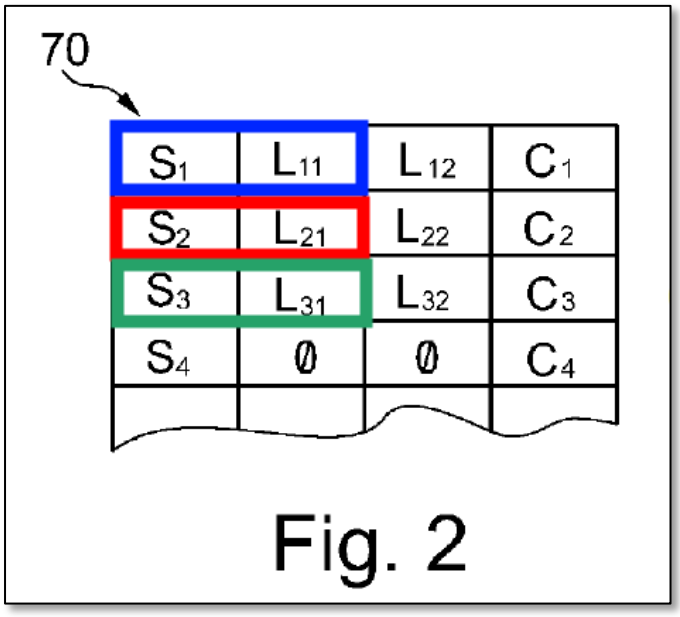
b. "the second web content class defining a second plurality of URLs different than the first plurality of URLs"

152. Specifically, Fages teaches that an administrator "gets connected to the server 20" and "uses this connection to define … different lists L_{i1} L_{i2} and C_i." Fages (EX1004), [0084]-[0086] ("Initially, at a step 80, an administrator of the system 2 gets connected to the server 20 by means of the station 24 and the network 22. The administrator uses this connection to define the table 70 as well as the different lists L_{i1} L_{i2} and C_i. After having defined the different possible Internet sessions through the server 20, the administrator records the sessions in the files 48.").

153. Fages teaches that the list of URLs L_{i1} defines "secure" or "sensitive" web content that may be accessed only via a respective "secure session," such as session S_1, S_2, or S_3. *Id.*, [0072]-[0074] ("the sessions S_1 to S_3 are secure sessions because they are each associated with a non-vacant list L_{i1}. Only the session S_4 is an unsecured session."), [0091]-[0093] ("an unsecured Internet session such as for example the session S_4, then a stage [90] of unsecured browsing is carried out on the Internet network 4. At the stage 90, the user can browse on all the Internet sites accessible on the Internet 4, except for those corresponding to URLs listed in the lists L_{i1}."), [0098]-[0100] ("At the step 112, whenever the user types out a new URL or is being directed towards a new URL, the new URL is compared with the lists L_{i1} and L_{i2} associated with the secured Internet session S_i selected. If the new URL belongs to one of the lists [L_{i1}] or L_{i2}, then the connection to this Internet site from this execution environment is authorized.], *id.* ("If the new URL belongs neither to

the list L_{i1} nor to the list L_{i1}, then the connection to the corresponding Internet site is not authorized.").

154. Fages teaches that secure session S_1 has a corresponding sensitive/secure URL list L_{11}, secure session S_2 has a corresponding sensitive/secure URL list L_{21}, and secure session S_3 has a corresponding sensitive/secure URL list L_{31}. *Id.*, [0072]-[0073], Fig. 2. Thus, each sensitive/secure URL list L_{i1} includes its own set of URLs accessible only during a secure session. Fages's annotated Fig. 2 below depicts secure session S_1 and URL list L_{11}, secure session S_2 and URL list L_{21}, and secure session S_3 and URL list L_{31}:

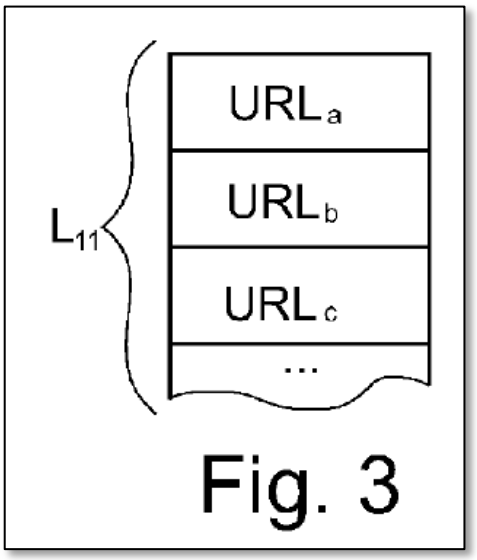


Id., Fig. 2 (annotated).

155. Fages teaches that its URL lists define a plurality of URLs. Fages teaches that list L_{11} "has at least three URLs respectively denoted as URL_a, URL_b,

and URL_c. These URLs correspond to URLs of sensitive Internet sites which have to be protected against attacks of the second type. These URLs are accessible only through the session S_1." *Id.*, [0073].

156. A POSITA would have understood that similarly, list L_{21} would be accessible only through session S_2, and likewise list L_{31} would be accessible only through session S_3. Indeed, list L_{11} being accessible only via secure session S_1 would have indicated to a POSITA that list L_{11} contains a plurality of secure/sensitive URLs that are different from the secure/sensitive URLs in lists L_{21} and L_{31}. Similarly, lists L_{21} and L_{31} would have been accessible only via their respective secure sessions S_2 and S_3, where each list has its own set of secure/sensitive URLs. The URLs in list L_{11} are depicted in Fages's Fig. 3 below:



Id., Fig. 3.

c. "the second web content class further being associated with sensitive information presenting a security risk"

157. Fages's data defining sensitive URLs in list L_{i1} are *associated with sensitive information presenting a security risk*, such as online banking websites. Fages (EX1004), [0056], [0088] ("a sensitive site such as an Internet site enabling the management of a user's bank accounts"). Indeed, for the same reasons the '721 Patent describes a URL request to www.bankofamerica.com as "sensitive" such that a new browser handles that request for "security reasons," Fages's URLs to banking account websites, handled by a separate browser instance, are "*associated with sensitive information presenting a security risk*" as claimed.

158. For example, the '721 Patent describes www.bankofamerica.com as being a sensitive website. '721 Patent (EX1001), 4:33-5:4 ("Referring to FIG. 1, a request is made to www.bankofamerica.com. Referring back to Fig. 6, the automatic FBI system determines if the URL is sensitive at 606 and if so, the automatic FBI system redirect the first browser instance to a default page at 608."), *id.* ("In the illustrated embodiment of FIG. 1, the message indicates that the requested website handles sensitive information and that a new browser window will be opened to handle the request for security reasons."), 7:65-8:11 ("The browser could not communicate directly with www.bank.com using the actual IP address of www.bank.com because that IP address of www.bank.com is not added to the whitelist of the current instance."), 3:7-11 ("In another embodiment, the automatic

FBI implements an effective approach for providing secure access to sensitive content for a novice user, while allowing an experienced user to specify COI classes of content that can coexist in a browser instance.").

159. *Compare* '721 Patent (EX1001), 4:65-5:4 ("Referring back to FIG. 1, when a URL is sensitive, instead of displaying the requested page in the current instance browser 102, the automatic FBI system displays a message at 104. In the illustrated embodiment of FIG. 1, the message indicates that the requested website handles sensitive information and that a new browser window will be opened to handle the request for security reasons."), Figs. 1-2 *with* Fages (EX1004), [0056] ("The Internet 4 is formed by numerous Internet servers connected to one another by means of a set 6 of long-distance information-transmission networks. This set 6 incorporates especially many routers so that it can route the information delivered by a server to any terminal connected to one of the networks of the set 6. By way of an illustration, only two Internet servers 8 and 10 have been shown. The server 8 is a sensitive Internet server. The term 'sensitive' designates, in this embodiment, a server that must be protected against second type attacks. For example, the Internet server 8 is an Internet server that enables online handling of bank accounts."), [0088] ("In response, at a step 86, the server 20 creates an execution environment and launches the execution in this environment of the Internet browser identified and configured as indicated in the list Ci associated with the identifier Si acquired during

the step 84. Thus, a sensitive site such as an Internet site enabling the management of a user's bank accounts will be accessible only by means of an Internet browser whose configuration defined in the list Ci is appropriate to this use. Thus, problems of compatibility between the Internet browsers and the Internet sites consulted are avoided.").

160. A POSITA would have understood that sensitive web content such as online handling of bank accounts would have presented a security risk because a successful attack or exploit would have allowed a hacker to illicitly access a user's account to steal personal information or possibly transfer funds to another account without the user's permission, for example. *See* §VI.C above.

d. "first web content class defining a first plurality of Uniform Resource Locators (URLs)"

161. Fages also teaches an administrator defines "unsecured" web content ("*first web content class*") that may be accessed only via an unsecured session S_4 because the "unsecured session enables connection with any Internet site except the sensitive Internet sites." Fages (EX1004), [0072]. A POSITA would have understood that the "sensitive Internet sites" in Fages refer to the sensitive/secure websites in lists L_{i1}. In one example, Fages teaches a user browsing in "an unsecured

Internet session such as for example the session S_4, then a stage [90][3] of unsecured browsing is carried out on the Internet network 4. At the stage 90, the user can browse on all the Internet sites accessible on the Internet 4, except for those corresponding to URLs listed in the lists L_{i1}." *Id.*, [0091]. Thus, a POSITA would have understood that Fages teaches a *first web content class that defines a first plurality of URLs* and that the first web content class would have defined a first plurality of URLs that would have included any URLs accessible via session S_4.

162. Thus, Fages teaches defining a *first plurality of URLs* that include the set of "all the Internet sites" not listed in lists L_{i1}. Fages further teaches that "at a step 92, whenever the user types out a new URL or whenever he is redirected to a new URL, for example following the selection of an Internet link, this new URL is compared with the URLs contained in all the lists L_{i1}. If a new URL does not belong to any of the lists L_{i1}, then the connection to this Internet site is authorized" and the

---

[3] A POSITA would have understood that Fages has a typographical error "stage 50," which I understand that Petitioner corrected to read "stage 90." Element 50 correctly refers to "computer 50," whereas element 90 refers to "stage 90" in the sentence immediately following "stage 50." Fages (EX1004), [0066] ("electronic computer 50"), [0070]-[0071] ("computer 50"), [0091] ("stage 90"), Fig. 1 (depicting computer 50), Fig. 6 (depicting stage 90).

Internet browser connects to the site at step 94. *Id.*, [0091]. However, if the requested URL is on a list L_{i1}, then "at a step 96, the connection to the Internet site corresponding to this URL is not authorized" which "therefore prevents the user getting connected to a sensitive Internet site during an unsecured Internet session" such as session S_4. *Id.*, [0092].

163. Thus, a POSITA would have understood that the *first web content class* (non-secure/non-sensitive) *defines a first plurality of URLs* which include the set of URLs not included in sensitive/secure lists L_{i1}, for example. A POSITA would have understood from Fages's teachings that this set of URLs for unsecure session S_4 would have defined a plurality of URLs because Fages teaches "any Internet site except the sensitive Internet sites" is included in the list of websites accessible via unsecure session S_4.

164. **In sum**, Fages teaches "receive data defining a first web content class [Fages's non-sensitive/non-secure web content] and a second web content class [Fages's sensitive/secure web content], the first web content class defining a first plurality of Uniform Resource Locators (URLs) [Fages's plurality of URLs related to non-sensitive/non-secure web content accessible via unsecure session S_4], and the second web content class defining a second plurality of URLs different than the first plurality of URLs [Fages's URL list L_{i1} that includes a plurality of URLs related to sensitive web content, where these URLs are different from the set of URLs

accessible via unsecure session S_4], the second web content class further being associated with sensitive information presenting a security risk [Fages's web content such as online banking]."

7. Claim 1(b): "generate a first browser instance, the first browser instance defining a browser instance type predetermined to be associated with the first web content class;"

a. '721 Patent and '721 File History "browser instance type"

165. Claim 1(b) recites a first browser instance type that is "*associated with the first web content class*." I note that the claims themselves do not further limit a browser instance/browser instance type (*e.g.*, Firefox vs. Chrome is not claimed). *See* Claims 1-18 (Table of Challenged Claims on page 10 above).

166. The '721 Patent specification describes browser type/browser instance type in relation to content classes, where one browser instance type handles "sensitive" content and another browser instance type is for "general" browsing. '721 Patent (EX1001), Abstract, 4:33-5:21 (describing the "novice user" embodiment where one fresh browser instance (FBI) is used to handle sensitive sites and a separate browser instance handles "all other sites."), 5:22-7:6 (describing the "experienced user" embodiment where a user may specify a browser such as Chrome for accessing certain websites and conflict of interest (COI) classes), 7:65-8:6, 8:38-56. The "novice user" embodiment thus does not specify using a particular browser

from a particular vendor/company for handling all the sensitive or general content, for example.

167. The ’721 Patent specification includes an example under the “experienced user” embodiment where Google Chrome is used for e-commerce and Mozilla Firefox is used for banking. *Id.*, 5:49-6:25. In this embodiment, different browsers from different companies/vendors are used for the different content classes. However, the ’721 Patent also describes “while Google Chrome and Firefox have been provided as examples, it is contemplated that any type of internet browser, or any browser accessible by a user, may be used.” *Id.*, 5:66-6:2. Thus, even the “experienced user” embodiment is not limited to using these specific browsers or using browsers from different companies/vendors to handle sensitive content vs. general content.

168. This understanding of browser instance/browser instance type comports with the ’721 FH. During prosecution the Applicant amended the claims to recite “generate a first browser instance, the first browser instance defining a browser instance type predetermined to be associated with the first web content class.” ’721 FH (EX1002), 280.

169. The Examiner associated browser instance type/browser type with sensitive content vs. general content where a private browser handles sensitive content and a standard/default browser handles general content. *Id.*, 287 (“web site

communication is handled by the user's standard or default web browser."), *id.* ("the private browser contacts the web site associated with the URL and waits for the remote server to return the requested web page to the private browser."), 287-288 ("If the IP address of the remote server matches one of the IP addresses associated with the web site URL in the web site database, the agent program retrieves the user's credentials from the password store and passes them to the private browser which in turn passes them to the web site."), 339-340.

170. Applicant did not argue that the Examiner's understanding was incorrect. Nor did Applicant amend the claims to further narrow browser instance/browser instance type to require different browsers from different companies/vendors.

b. Fages's "first browser instance" and "browser instance type"

171. As detailed next, Fages teaches the same understanding of "browser instance"/"browser instance type" as the '721 Patent. Fages teaches generating a first browser session/instance ("*first browser instance*") in an execution environment. Fages (EX1004), [0003] ("an access server connected to the Internet and capable of the parallel execution of **several instances of an Internet browser**"), [0018]-[0020] ("An object of the invention therefore is a method for browsing on the Internet wherein: if the URL belongs to a pre-recorded list of URLs of sensitive Internet sites to which connection from the first execution environment is not

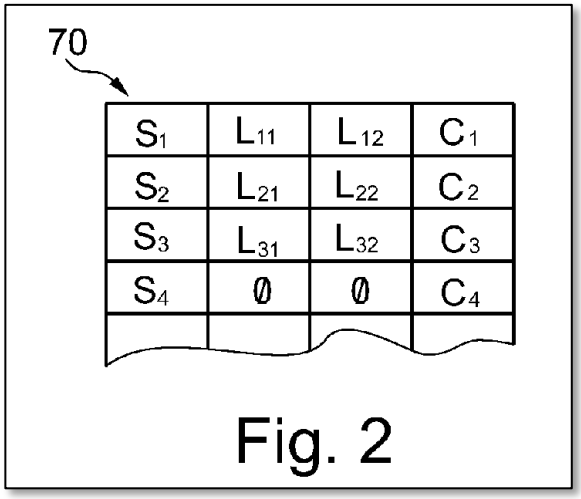
authorized, then connection to this Internet site is authorized only on the part of the remote control of a second Internet browser executed in a second execution environment on the access server so that this second Internet browser is used, instead of the first Internet browser, to connect to the sensitive Internet site identified by its URL, and if not, the connection is made to the Internet site identified by this URL using the first Internet browser."), [0044]-[0048] ("An object of the invention is also an information-recording medium comprising instructions for executing the above browsing method when these instructions are executed by an electronic computer. An object of the invention is also an access server to implement the above browsing method, this server being capable of the parallel execution of several Internet browsers in execution environments that are independent of one another so that a user station can take remote control over an Internet browser executed in a first execution environment on the access server so that this first Internet browser connects to an Internet site identified by its URL. This access server is also capable: if the URL belongs to a pre-recorded list of URLs of sensitive Internet sites to which connection from the first execution environment is not authorized, the connection to this Internet site is authorized only on the part of the remote control of a second Internet browser executed in a second execution environment on the access server so that this second Internet browser is used instead of the first Internet browser to connect to the sensitive Internet site identified by this URL, and capable if not, of

getting connected to the Internet site identified by this URL by means of the first Internet browser. Finally, an object of the invention is also a user station for the implementing of the above method for browsing, this user station being equipped with a software program for taking remote control of an Internet browser executed on the above access server when the user station is connected, for this purpose, to the access server by means of an information-transmission network.").

172. A POSITA would have understood that Fages's browser sessions are each a *browser instance* at least because Fages uses the terms synonymously and interchangeably. *Id.*, [0003] ("several instances of an Internet browser"), [0072] (table 70 shows "list of Internet sessions configured on the server 20."), [0072] ("Here, the first column of this table lists identifiers S_i of different Internet sessions which the user of the station 26 can launch."), [0104], [0072] ("Only the session S_4 is an unsecured session. This unsecured session enables connection with any Internet site except the sensitive Internet sites."); [0073] ("These URLs are accessible only through the session S_1."), [0104].

173. A POSITA would have understood that Fages teaches different browser instances because each session is utilized to connect to a particular URL in a particular browsing execution environment at a time, for example. *Id.*, [0072] ("FIG. 2 shows, in the form of a table 70, a list of Internet sessions configured on the server 20. An Internet session is a set of pieces of information used by the server 20 to

configure and manage the use of an Internet browser in an execution environment dedicated to it."). A list of Internet sessions is depicted below:



Id., Fig. 2.

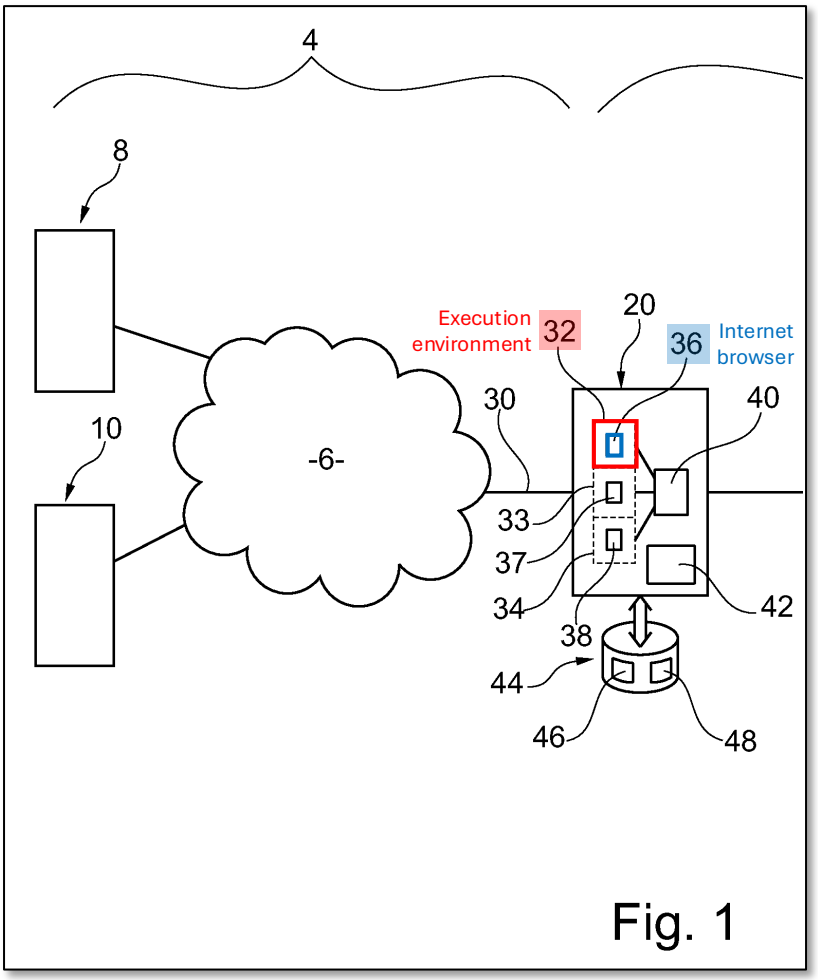
174. I also note that Fages uses phrasing including "launched the execution of the Internet browser." *Id.*, [0089]. Thus, a POSITA would have understood that Fages thus teaches generating at least a first browser instance.

175. Fages further teaches that the first browser session/instance defines a *first browser instance type* (*e.g.*, non-sensitive) predetermined to be associated with the first web content class (*e.g.*, non-sensitive content in URL). *Id.*, [0001]-[0003], [0018]-[0020]. "Sensitive"/"non-sensitive" and "non-secure" are "browser instance types" consistent with the '721 Patent's specification. *See* '721 Patent (EX1001), 4:35-37 ("the FBI system enables the processing of two types of sites for a novice user: sensitive sites and every other type of site"). A POSITA would have understood that when users are browsing sensitive vs. other types of web sites, the

users are therefore browsing different instances (or classes) of content: secure vs. non-secure.

176. When a user while in a non-secure browsing session S_4 tries to access a URL not on the predetermined lists of sensitive/secure URLs, a connection is made to the Internet site using the first Internet browser (*e.g.*, browser 36 of execution environment 32). Fages (EX1004), [0018]-[0020], [0091]-[0092]. *See* Claim 1(Pre) (§VII.A.5 above).

177. In more detail, Fages teaches a first browser instance used for session S_4 regarding non-secure web content (*e.g.*, non-sensitive URLs). Fages (EX1004), [0018]-[0020], [0044]-[0046], [0072]-[0074], [0084]-[0093] ("If … the identifier S_i acquired corresponds to an unsecured Internet session such as for example the session S_4, then a stage 50 of unsecured browsing is carried out on the Internet network 4. At the stage 90, the user can browse on all the Internet sites accessible on the Internet 4, except for those corresponding to URLs listed in the lists L_{i1}."). As depicted in Fages's annotated Fig. 1 below, Internet browser 36 is executed in execution environment 32:



Id., Fig. 1 (excerpt, annotated).

178. Fages provides details of server 20. *Id.*, [0063] ("The server 20 is connected to the Internet 4 by means of an information-transmission link 30. It is capable of simultaneously executing several Internet browsers in distinct execution environments. For example, in FIG. 1, three distinct execution environments 32, 33 and 34 are schematically represented by rectangles of dashes. The Internet browsers executed in each of these environments 32 to 34 are represented by squares 36, 37 and 38 in unbroken lines."), [0089]-[0093].

179. A POSITA would have understood that the other browser execution environments 33 and 34 (along with browsers 37 and 38) would likewise have been utilized to access URLs, and that Fages teaches partitioning browsers into separate execution environments so sensitive/secure URLs are not accessed by a browser accessing non-sensitive/non-secure URLs. *Id.*, [0104] ("The method described here obliges the user to partition the secured Internet sessions from the unsecured Internet sessions. This partitioning is done by means of Internet browsers executed in different execution environments. As a result, even if the user connects simultaneously to a malicious Internet site and to a sensitive Internet site, the malicious Internet site cannot affect the working of the sensitive Internet site since the Internet browsers used to connect to these two sites are executed in execution environments that are independent of one another.").

180. In other words, Fages teaches that each of its execution environments utilize browsers for sensitive/secure and non-sensitive/non-secure URLs, but browsers are "partitioned" such that a browser in a single environment (*e.g.*, environment 32) does not access both sensitive/secure and non-sensitive/non-secure URLs at the same time or in a single session. For example, URLs from secure/sensitive URL list L_{11} may be accessed only via secure session S_1 executing in any of environments 32-34, and URLs not in any secure URL list L_{i1} may be

accessed by non-secure session S_4 executing in either of the execution environments not being utilized for session S_1.

181. Moreover, a POSITA would have understood that Fages teaches utilizing multiple independent browser execution environments to allow for a user to have multiple concurrent browsing sessions open. Similarly, using multiple independent browser execution environments each with its own browser instance would have allowed for both sensitive and non-sensitive URLs to be browsed concurrently while also maintaining security to not allow a sensitive URL to be accessed via a non-secure session/instance, which Fages refers to as "partitioning."

182. Lastly, a POSITA would have understood that separate browser instances/sessions (*e.g.*, S_1-S_4) would be utilized for each browser in each browser execution environment. For example, Fages teaches utilizing sessions S_1-S_3 to handle sensitive/secure web content and teaches using session S_4 to handle the non-sensitive/non-secure web content. *Id.*, [0072] ("FIG. 2 shows, in the form of a table 70, a list of Internet sessions configured on the server 20. An Internet session is a set of pieces of information used by the server 20 to configure and manage the use of an Internet browser in an execution environment dedicated to it. This table 70 is a part for example of the configuration files 48. Here, the first column of this table lists identifiers S_i of different Internet sessions which the user of the station 26 can launch. The second column of the table 70 associates, with each of the identifiers

S_i, a pre-recorded list L_{i1} of URLs of sensitive Internet sites that must be protected against attacks of the second type. The third column contains, for each session S_i, a second list L_{i2} of URLs of authorized Internet sites and, if necessary, accessible from other sessions. Finally, the fourth column contains, for each session S_i a list C_i of configuration parameters of the Internet browser. Among the four sessions S_1 to S_4 represented here, the sessions S_1 to S_3 are secure sessions because they are each associated with a non-vacant list L_{i1}. Only the session S_4 is an unsecured session. This unsecured session enables connection with any Internet site except the sensitive Internet sites. In particular, the session S_4 enables the user to get connected as the case may be with malicious Internet sites such as those hosted by the server 10. To this end, the lists L_{41} and L_{42} are non-existent and represented by the symbol φ.”). Thus, a browser execution environment being utilized for non-sensitive web content would be using a browser instance/session S4 and a separate browser execution environment would be utilized to handle a session for sensitive web content (e.g., any one of S_1-S_3 that is available). *Id.*, [0072].

183. **In sum**, Fages teaches “generate a first browser instance [Fages’s session S_4 on a browser such as 36 in a browser execution environment such as 32], the first browser instance defining a browser instance type [Fages’s instance for non-sensitive web content] predetermined to be associated with the first web content

class [Fages's session S_4 is predetermined to be used only for the class of non-sensitive web content]."

8. Claim 1(c): "intercept a request to access web content from the first browser instance, the request defining a URL of the second plurality of URLs such that the web content is associated with the second web content class;"

184. Fages teaches that Internet browser instance 36 in execution environment 32 accesses content from the plurality of URLs of non-sensitive content and "a stage [90] of unsecured browsing is carried out on the Internet network 4." Fages (EX1004), [0088]-[0091]. When a user submits a request for a new URL from the first browser instance while browsing in non-secure session S_4, "this new URL is compared with the URLs contained in all the lists L_{i1}." *Id.*, [0091]. Fages teaches that if the requested URL is on the list of sensitive/secure URLs, then the access to this sensitive/secure URL is blocked for Internet browser 36 of execution environment 32 of session S_4 because an "unsecured session enables connection with any Internet site except the sensitive Internet sites." *Id.*, [0088]-[0089], [0091]-[0092], [0072]-[0074]. Another browser instance may be utilized to access a sensitive URL, as detailed in Claim 1(e). *See* §VII.A.14 below, Claim 1(e); *see also* Fages (EX1004), [0098]-[0099], [0109]-[0110]. Otherwise, if the URL is non-sensitive (*i.e.*, not contained in a sensitive URL list), "then the connection to this Internet site is authorized" and the URL is accessed, for example, using browser 36 in execution environment 32 and session S_4. Fages (EX1004), [0090]-[0093].

185. Continuing further with the example, a POSITA would have understood that in the aforementioned example, if a user attempted to access sensitive web content using browser 36 in browser execution environment 32, a new browser (*e.g.*, 37) would need to be utilized in another browser execution environment (*e.g.*, 33) to allow the sensitive/secure content to be viewed (assuming the URL is on the URL list of sensitive web content). This would have been in keeping with Fages's stated goal of partitioning web browser instances to prevent sensitive web content from being accessed in a browser that is accessing non-sensitive web content. *Id.*, [0104], [0109]-[0110], [0072]-[0074].

186. Otherwise, if a user is already utilizing browser 36 in browser execution environment 32 to access non-sensitive/non-secure content, and a user tries to access a URL on the non-sensitive URL list, "then the connection to this Internet site is authorized" and the non-sensitive URL is accessed, for example, using that same browser 36 in execution environment 32 and session S_4. *Id.*, [0090]-[0093].

187. Thus, a POSITA would have understood that Fages teaches that server module 40 intercepts requests to access sensitive/secure web content that are sent to server 20 from client module 52 (*i.e., intercept a request to access web content from the first browser instance, the request defining a URL of the second plurality of URLs such that the web content is associated with the second web content class*). *Id.*, [0072]-[0074], [0086]-[0088], [0090]-[0093], [0098]-[0099], [0109].

188. In one embodiment, the sensitive URL will automatically be sent to execute in a different session (*e.g.*, session S_1-S_3) of a different browser environment (*e.g.*, browser execution environment 33 or 34) rather than a browsing environment being used for non-secure browsing (*e.g.*, browser execution environment 32) of session S_4 instead of simply blocking the attempt. *Id.*, [0092] ("If not, at a step 96, the connection to the Internet site corresponding to this URL is not authorized. As a result, the connection to this Internet site is not set up. This therefore prevents the user getting connected to a sensitive Internet site during an unsecured Internet session. Thus, a simultaneous connecting [sic] with a malicious Internet site and a sensitive Internet site from an Internet browser executed in the same execution environment is automatically made impossible."), [0109]-[0110] ("As a variant, rather than blocking a URL with which it is not possible to get connected through the Internet browser executed in the current execution environment, the server 20 automatically launches the execution of another Internet browser in another execution environment from which the connection to this URL is possible and the remote control of the second Internet browser automatically replaces the remote control of the first Internet browser. When several Internet sessions are executed simultaneously, they are displayed in distinct windows on the screen 58, or, on the contrary, in one and the same window. In the latter case, the different Internet

sessions executed simultaneously are displayed, for example, in different tabs of a same window.").

189. In either case, server 20 including server module 40 that intercepts the request to access web content and either allows, blocks, or opens a new browser instance in another browsing environment in response to the request, based on the browser execution environment and list settings. A POSITA would have understood that the requests would have been intercepted because a user is not allowed to simply browse to a sensitive URL without the system monitoring, inspecting, and allowing such action to occur.

190. While Fages also teaches simply blocking a user's attempts to access a sensitive URL, in one embodiment, the sensitive URL will automatically be sent to execute in a different browser environment (*e.g.*, browser execution environment 33 or 34) rather than an unsecured browsing environment (*e.g.*, browser execution environment 32) instead of simply blocking the attempt. *Id.*, [0092], [0109]-[0110]. In either case, server 20 including server module 40 intercepts the request to access web content and either allows, blocks, or opens a new browser in response to the request based on the browser execution environment and list settings. As I opined in Claims 1(a) and 1(b) (§§VII.A.6–VII.A.7 above) , this process is in line with Fages's express goal of maintaining partitioning between sensitive/secure and non-

sensitive/non-secure URLs in different browsers in different browser execution environments.

191. **In sum**, Fages teaches "intercept a request to access web content from the first browser instance, the request defining a URL of the second plurality of URLs [Fages's sensitive URL in list L_{i1}] such that the web content is associated with the second web content class [Fages's sensitive web content class]."

9. Claim 1(d): "confirm that the URL is not being accessed by another separate already-running browser instance associated with the second web content class, by"

a. Fages's Teachings

192. Fages teaches multiple browser instances and browser execution environments and "prevent[ing] the user getting connected to a sensitive Internet site during an unsecured Internet session." Fages (EX1004), [0063], [0072]-[0073], [0088]-[0092], [0104], [0109]-[0110], Fig. 1. *See* §§VII.A.6–VII.A.8 above, Claim limitations 1(a)-(c). Fages teaches launching non-secure/non-sensitive URLs in browser 36, for example, and launching secured/sensitive URLs in a different browser instance (*e.g.*, 37 or 38). Fages (EX1004), [0044]-[0046], [0109]-[0110]. As I opined in §VII.A.8 above, a POSITA would have understood that Fages would have utilized different browsers in different browser execution environments to handle sensitive content vs. non-sensitive content. *See* §VII.A.8 above, Claim 1(c).

193. More specifically, a POSITA would have understood that Fages teaches utilizing separate browser execution environments and separate browsers such that sensitive URLs are not accessed using a browser that accesses non-sensitive URLs, and therefore "the malicious Internet site cannot affect the working of the sensitive Internet site since the Internet browsers used to connect to these two sites are executed in execution environments that are independent of one another." Fages (EX1004), [0044]-[0046], [0018]-[0021], [0104], [0109]-[0110]. This understanding is in line with Fages's teachings on partitioning sensitive web content in one session/instance from non-sensitive content in another session/instance. However, Fages does not expressly discuss *confirming that the URL is not being accessed by another separate already-running browser instance associated with the second web content class*. That is, Fages does not expressly discuss checking to see if a session is already open for viewing sensitive content. Jai teaches comparing a requested URL with open URLs. Hayden teaches confirming if a browser instance is launched and available to handle a requested URL.

b. Jai's URL-Comparison Technique

194. Jai teaches a computer system that performs a URL comparison to detect if a requested URL is already open in a browser instance. Jai (EX1006), 3:22-4:2, 4:20-21 ("Figure 8 is a schematic diagram of a web browser illustrating a web site having a minimized application of a second website"), 12:8-13:7 ("Another

aspect of the present invention is illustrated in Figure 8 and prevents redundant applications of the same website 20 from being opened at a given time. When a user inputs a URL and requests that the associated site by opened, a search is made of the other loaded websites 20. If a loaded site having the same URL is already open, it will be displayed to the user without a second application being opened."), 14:3-5 ("Many of the examples and illustrations describe and show a single embodiment. However, two or more applications of the invention may be used in a single device."), Fig. 8. Jai teaches that in this way "duplicate applications of the same site may also be avoided." Jai (EX1006), 3:22, Abstract. This is because "prior to accessing and loading another site, the present invention searches other open applications and determines whether the new request is actually a duplicate of an already open application. If it is a duplicate, the existing application is displayed and the second requested application of the same site is deleted." Jai (EX1006), 3:22-4:2.

195. Specifically, Jai's Fig. 8 "is a schematic diagram of a web browser illustrating a web site having a minimized application of a second website." Jai, 4:20-21. Jai's Fig. 8 annotated below depicts address box 36 (for typing in a URL), website 20 (website a user is currently browsing), location 83 (a "currently loaded website," specifically www.ericsson.com, which is the same website 20 the user is

currently browsing), and location 81 (another "currently loaded website" that is "loaded and has been minimized," specifically www.cellularphones.com):

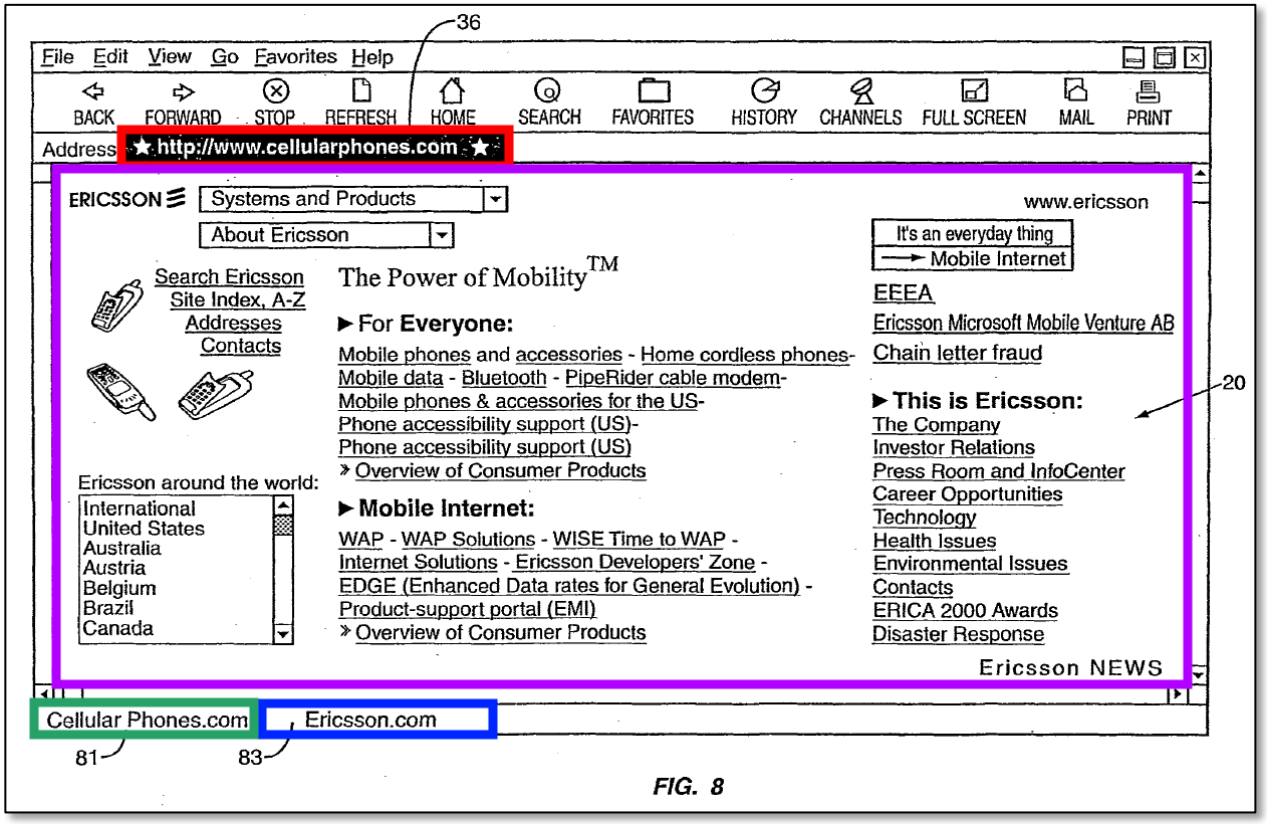


FIG. 8

Jai (EX1006), Fig. 8 (annotated), 12:14-18.

196. Jai teaches that an "aspect of the present invention is illustrated in Figure 8 and prevents redundant applications of the same website 20 from being opened at a given time. When a user inputs a URL and requests that the associated site b[e] opened, a search is made of the other loaded websites 20. If a loaded site having the same URL is already open, it will be displayed to the user without a second application being opened." Jai (EX1006), 12:8-14, 12:17-13:2 ("When the user enters the requested URL, the present invention will check the other loaded

applications. Upon finding that another application of cellularphones.com is already open, this website [www.cellularphones.com] will be displayed in website 20 without opening a redundant application of cellularphones.com.").

c. Hayden's Teachings – "*another separate already-running browser instance*"

197. While the Fages-Jai combination teaches checking if the requested URL is already open, it does not discuss checking other running browser instances. Hayden describes how to determine if a particular browser instance is already launched, and if a particular browser instance is not already launched, Hayden teaches launching the particular browser instance. Hayden (EX1050), 35:4-36:32, Fig. 13 (reproduced and annotated just below). Specifically, Hayden describes how "FIG. 13 illustrates an example routine 1300 implemented by a virtual machine manager 182 or other component of the network computing provider 107 for routing requests to browser instances that have been customized to handle the request." Hayden (EX1050), 35:4-8. Moreover, "if a user of a client computing device 102 submits a request for http://www.xyzwebsite.com, the instance selection module 196 of the virtual machine manager 182 can determine which of the available network computing components to route the request to, based on whether a customized browser virtual machine image 186 has been created for the requested site and, if so, whether a browser has been instantiated from the image 186." Hayden (EX1050), 35:8-15.

198. Hayden teaches that at block 1306, "if a browser virtual machine image 186 containing a customized browser corresponding to the requested web site has already been created, the routine 1300 can proceed to decision block 1308." Hayden (EX1050), 35:38-62. Hayden teaches that at "decision block 1308, the instance selection module 196 can determine whether a browser, customized for the requested web site, has been instantiated from a browser virtual machine image 186 and is ready to process a new request." Hayden (EX1050), 36:7-11. Hayden teaches that "in response to receiving the request, the instance selection module 196 can consult a database, query a service, or perform some other lookup of currently running customized browser instances to determine whether a browser customized for the http://www.xyzwebsite.com" "has been instantiated and is available." Hayden (EX1050), 36:11-18. Hayden teaches that if a browser customized for the requested website "has been instantiated and is available," then "the routine can proceed to block 1314. If none is available, the routine 1300 can proceed to block 1312." Hayden (EX1050), 36:18-20.

199. Hayden teaches that "at block 1314, the instance selection module 196 can route the request to the customized browser for processing." Hayden (EX1050), 36:26-32. Otherwise, if no customized browser instance was available at block 1308, then at block 1312, "the instance selection module 192 or another component of virtual machine manager 182 or the network computing provider 107 can launch

a customized browser from a browser virtual machine image 186 to process the request." Hayden (EX1050), 36:19-32. Afterwards, the process proceeds to block 1314. *Id*. Hayden's annotated Fig. 13 below depicts this process:

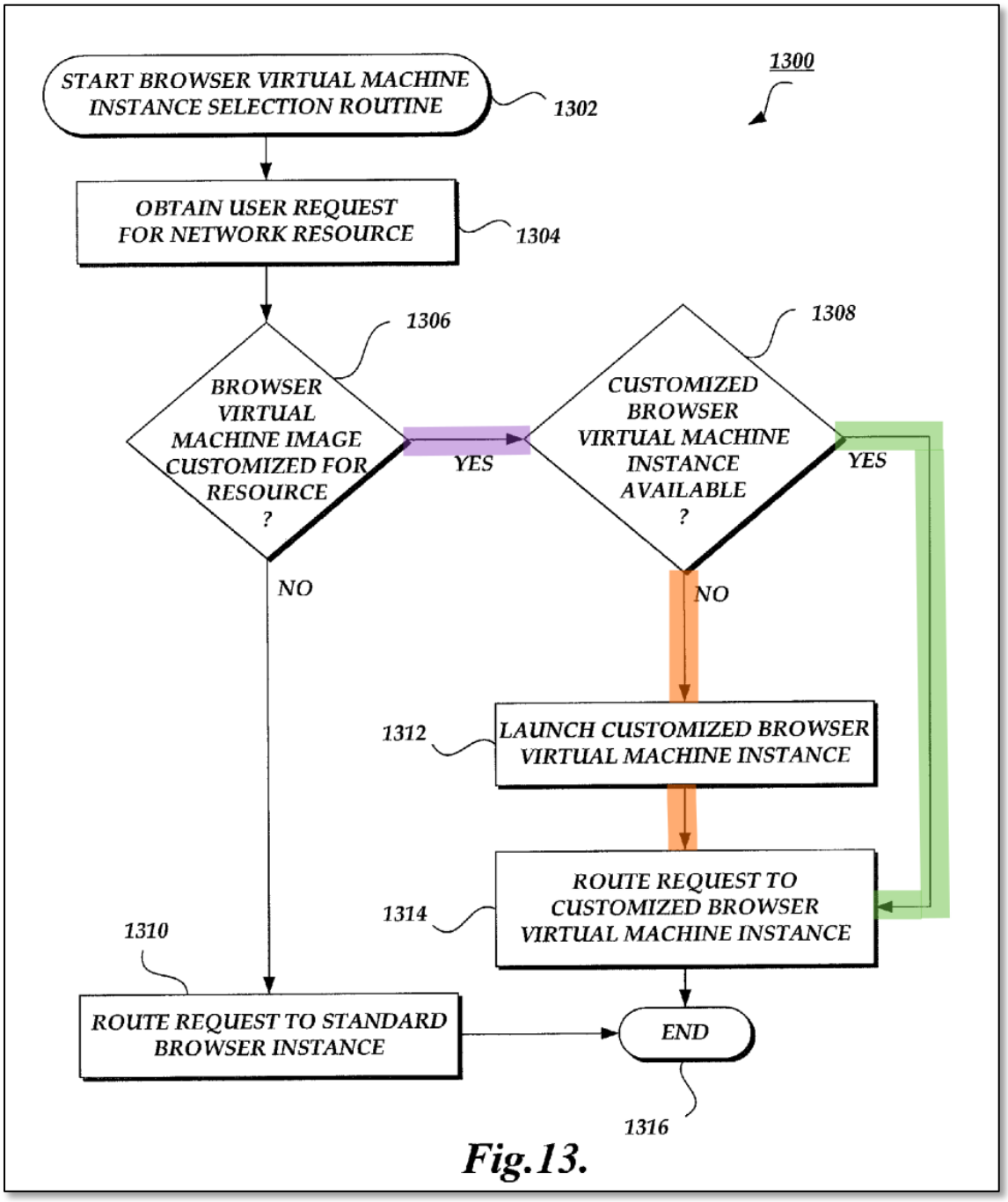


Fig.13.

Hayden (EX1050), Fig. 13, *see also id.*, 35:4-62, 7:32-43, 7:58-8:17, 8:45-9:1. Hayden's annotated Fig. 13 above depicts block 1306 and a "yes" result, block 1308 with a "yes" result leading to block 1314, and block 1308 with a "no" result leading to block 1312 to launch a customized browser virtual machine instance and proceeding to block 1314.

d. The Fages-Korycki-Jai-Hayden Combination Teaches "confirm that the URL is not being accessed by another separate already-running browser instance…"

200. Hayden's process confirms whether "*another separate already-running browser instance*" is available, and if one exists but is not currently running, Hayden's technique launches an instance. Similarly, Jai's URL-comparison process confirms that a URL is not being accessed because Jai compares a requested URL against the open/active URLs. It would have been obvious to modify the Fages combination to include URL and browser instance comparison steps to check the requested URL against all open browser instances (*e.g.*, browser instances in every one of Fages's execution environments). Therefore, the Fages-Korycki-Jai-Hayden combination confirms which browser instances and URLs within those browser instances are already open and therefore confirms the requested URL is *not* already open before partitioning a new URL to a specific browser instance, teaching the negative limitation of *confirming that the URL is **not** being accessed by another separately already-running browser instance*. The Fages-Korycki-Jai-Hayden

combination would have included URL and browser instance comparison steps to check the requested URL against all open browser instances (*e.g.*, browser instances in *every one of Fages's* execution environments).

201. In the proposed combination, Fages teaches partitioning sensitive and non-sensitive URLs in different browser instances. *See* Claim 1(a) §VII.A.6 above. Modifying Fages's native partitioning functionality with Jai's URL-comparison functionality and Hayden's browser instance confirming functionality allows Fages's system to check whether an active browser instance and URL is available with the requested URL before opening a duplicative instance. I note that this process simply adds step(s) to Fages's process by comparing a requested URL with (1) open URLs and (2) open browser instances for handling the URL, thus confirming whether or not a URL is currently open and whether a browser instance to handle a particular URL is already open before taking the step of partitioning a browser to open a requested URL in a new instance. For example, if a user requested to launch sensitive URL_a, the Fages-Jai-Hayden combination would confirm whether URL_a is already open in secure sessions S_1-S_3 before partitioning URL_a to a secure browser instance. In the case where the requested URL is already open, there would be no need to launch a new browser instance.

202. A POSITA would have understood that a currently open URL would already be in the correct secure or non-secure session because the Fages-Jai-Hayden

combination teaches checking URL lists before allowing a URL to be accessed. Thus, the system would simply allow the user to access the requested URL in the already-open browser instance.

203. In more detail, Fages would utilize Jai's URL-comparison functionality and Hayden's browser detection functionality after a user requests to access a URL. For example, at [0086]-[0087], Fages teaches module 52 of the client computer "acquires the identifier S_i of the session selected by the user of the station 26 and transmits this identifier to the module 40." Then, in response, "server 20 creates an execution environment and launches the execution in this environment of the Internet browser identified and configured as indicated in the list C_i associated with the identifier S_i acquired during the step 84." At [0088]-[0093], Fages teaches a scenario where a user has initially selected unsecure session S_4 and the user tries to access a sensitive URL. As a result, at step 96 "the connection to the Internet site corresponding to this URL is not authorized." Fages (EX1004), [0088]-[0093]. However, Fages teaches that "rather than blocking a URL with which it is not possible to get connected through the Internet browser executed in the current execution environment, the server 20 automatically launches the execution of another Internet browser in another execution environment from which the connection to this URL is possible and the remote control of the second Internet

browser automatically replaces the remote control of the first Internet browser." Fages (EX1004), [0109].

204. In a different scenario, Fages teaches a user initially selects a secure session (e.g., S_1-S_3) and tries to access a sensitive URL. This action is permitted because the requested sensitive URL was compared with the URL lists and was determined to be allowed to be accessed in the current session of the execution environment. Fages (EX1004), [0098]-[0100]. This scenario does not require a new browser instance to be launched because the current instance is suitable to handle the requested URL.

205. In the Fages-Jai-Hayden combination, an additional step (or steps) is added to Fages's process. Namely, Jai's teachings of determining what URLs are currently open and Hayden's teachings of determining whether a browser instance able to handle the requested URL is currently open is performed before a new browser instance is launched (Fages (EX1004), [0085]-[0093]) or an already existing browser instance is utilized (Fages (EX1004), [0098]-[0100]). In an example where a user requests sensitive URL_a, the Fages-Jai-Hayden combination compares the requested sensitive URL_a against all the URLs that are currently open (based on Jai's teachings) and all the browser instances that are currently open (Hayden's teachings). *See* §VII.A.9.b above. The Fages-Jai-Hayden combination would thus confirm that sensitive URL_a is not already open (i.e., *confirming that the*

URL is not being accessed by another separate already-running browser instance…) when URL_a is not among the set of all the currently open URLs in the currently open browser instances. A POSITA would have understood that if a requested URL is not among the set of all active URLs, then the requested URL is not being accessed in any of the open browser instances.

206. As a result, if the user is using unsecured/non-sensitive session S_4 and requests to access sensitive URL_a, the Fages combination would then check to see if URL_a is already open in a secure session, and if not, launch a new secure browser instance, one of sessions S_1-S_3, to open sensitive URL_a. *See* Claim 1(e) §VII.A.14. Because the Fages-Jai-Hayden combination detects all open URLs and browser instances and confirms the requested URL is not already open before partitioning a new URL to a specific browser instance, a POSITA would have understood that the Fages-Jai-Hayden combination teaches the negative limitation of ***confirming that the URL is <u>not</u> being accessed*** *by another separately already-running browser instance*.

207. While not expressly required by the claim language, I discuss in §VII.A.9.e below some implementation details for possible ways that a POSITA would have understood Fages would have been combined with Jai and Hayden. A POSITA would have understood that implementing URL-comparison functionality such as Jai's and browser instance detection functionality such as Hayden's in

Fages's computer system would have been achieved by various possible options, such as implementing URL-comparison functionality as a "stand-alone" part of a controller (*e.g.*, processor) where the URL-comparison functionality is "stand-alone" because the functionality is not part of a web browser. In this implementation, Fages's or the Fages-Korycki's processor would have been modified to additionally perform URL-comparison functionality, such as that taught by Jai. In the same manner, Fages would be modified to implement Hayden's browser instance detection functionality.

208. **In sum**, the Fages-Korycki-Jai-Hayden combination teaches "confirm that the URL [Fages's sensitive content URL] is not being accessed by another separate already-running browser instance [Jai's teachings of URL-comparison functionality and Hayden's browser instance detection functionality applied to Fages] associated with the second web content class [Fages's sensitive content class], by."

e. Motivation to Combine Fages with Jai and Hayden (Browser Instance Detection and Active URL Comparison)

209. A POSITA would have been motivated to modify Fages to include a URL comparison or similar functionality as taught by Jai, and browser instance detection or similar functionality as taught by Hayden, to advantageously detect all open browser instances and compare the open browser instances to a requested URL

to determine if another browser instance should be opened for the requested URL, for at least the following at least four reasons.

210. **First**, as I opine in §VII.A.9 above adding Hayden's browser instance detection functionality and Jai's URL-comparison functionality to Fages would have advantageously allowed the modified Fages to detect whether browsers and/or URLs are open or not by detecting the full set of all browsers and URLs that are currently open. Fages natively teaches opening a session/browser instance for sensitive content when a user attempts to access sensitive content from a non-sensitive session/browser instance. That is, Fages teaches separate execution environments and separate browsers and partitioning the execution environments so that sensitive/secure content is not accessed from a non-sensitive/non-secure browser instance. Fages (EX1004), [0063], [0072]-[0074], [0104], [0109]-[0110], [0019]-[0021]. Thus, Fages teaches partitioning non-sensitive content and sensitive content and opening a new browser instance when a user tries to access sensitive content from a non-sensitive session/browser instance.

211. A POSITA would have realized that since Fages partitions sensitive vs. non-sensitive content, together they comprise the full set of all possible content (*i.e.*, all content must be partitioned into exactly one of these two classes). Since Fages is concerned with security, it would have defeated its express purpose if, when users try to browse a new arbitrary URL, Fages could not tell if that new URL should fall

into the sensitive or non-sensitive classes. If Fages could not determine this, it might incorrectly place a sensitive URL in the non-sensitive class and vice versa.

212. Moreover, even if Fages could tell if a new URL belongs to the sensitive or non-sensitive classes, Fages would have had to find out if that URL had been seen before and is it open already or not. If a new URL is already open in another browser instance, Fages would have considered it a form of (well known) caching, and simply reuse the existing, already open browser instance. This would have permitted users, for example, to rejoin an existing sensitive banking or shopping session. If Fages did not detect if that URL was already open, it would have inadvertently re-opened new instances for the same URL repeatedly, losing session state (*e.g.*, online shopping cart) while wasting memory and computational resources with unnecessary duplicate browser instances.

213. **Second**, modifying Fages with Hayden's and Jai's teachings would have allowed Fages to determine which browser instances are currently open and/or closed to more efficiently manage execution environments and browser instances, such as by using this information to determine if a current sensitive/secure browser instance is already running and may be utilized. For example, if session S_1 is active and a user selects a URL in secure/sensitive URL list L_{11}, then the browser instance for session S_1 may be utilized to handle that URL. Conversely, if session S_4 is active and a user selects a URL in secure/sensitive URL list L_{11}, then the browser instance

for session S_4 would not be utilized to handle that URL, and a new session S_1 would need to be launched in a different execution environment.

214. **Third**, a POSITA would have understood that this modification would have reduced computing resource utilization by not having to launch a separate browser instance for each sensitive URL that a user wishes to launch because a currently running sensitive browser instance may be utilized instead (at least in some scenarios). Uola (EX1038), [0006] ("[R]unning many browser processes concurrently consumes much memory which may cause some problems to the capability of the device to handle each process[] efficiently enough. Each browser process may allocate even several megabytes of memory … depending on the complexity of the web site that the browser is showing.").

215. Thus, Fages's modified system would have been able to determine if no sensitive/secure browser instance is currently running/available and that a new sensitive/secure browser instance should be initiated or if a currently running sensitive browser instance was available for use instead of simply opening a new sensitive browser instance each time a sensitive URL is requested.

216. **Fourth**, this use of a known technique (a form of caching) would have improved Fages's similar computer system in the same way as Hayden's computer system and Jai's computer system by utilizing browser instance detection functionality (*e.g.*, implemented in programming instructions) to detect all the

browser instances that are open and URL-comparison/detection functionality (*e.g.*, implemented in programming instructions) to confirm if a requested URL is currently open or not, which would have logically allowed for the system to determine if a sensitive browser instance and/or URL is not already open. Combining prior art elements such as browser instance detection functionality and URL-comparison functionality with Fages's computer system elements according to known methods would have yielded the predictable result of a functional and improved Fages computer system. A POSITA would have understood that adding program instructions to Fages or modifying Fages's existing programming instructions would have been sufficient to implement this modification.

217. A POSITA would have had a reasonable expectation of success modifying Fages for at least six reasons.

218. **First**, as I discuss in §VII.A.9.e above, the modification would have required only straightforward changes to programming instructions, and browser instance detection functionality and URL-comparison functionality can be implemented alongside a processor; a POSITA would have understood this to be achievable by utilizing the processor in the Fages or Fages-Korycki system.

219. **Second**, Fages, Jai, and Hayden teach similar computer systems, such that Fages's hardware would not have needed to be modified to implement the Jai

and Hayden modifications. For example, compare Fig. 1 of Fages with Fig. 1B of Hayden below:

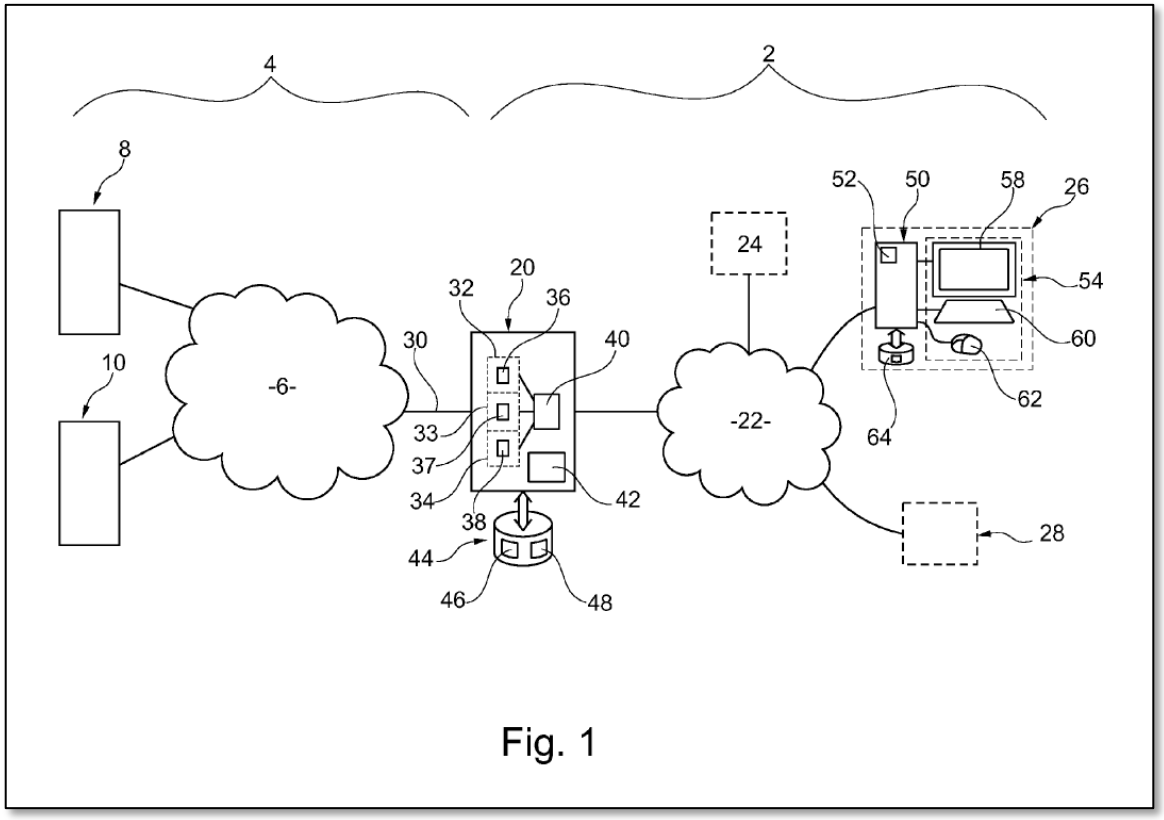
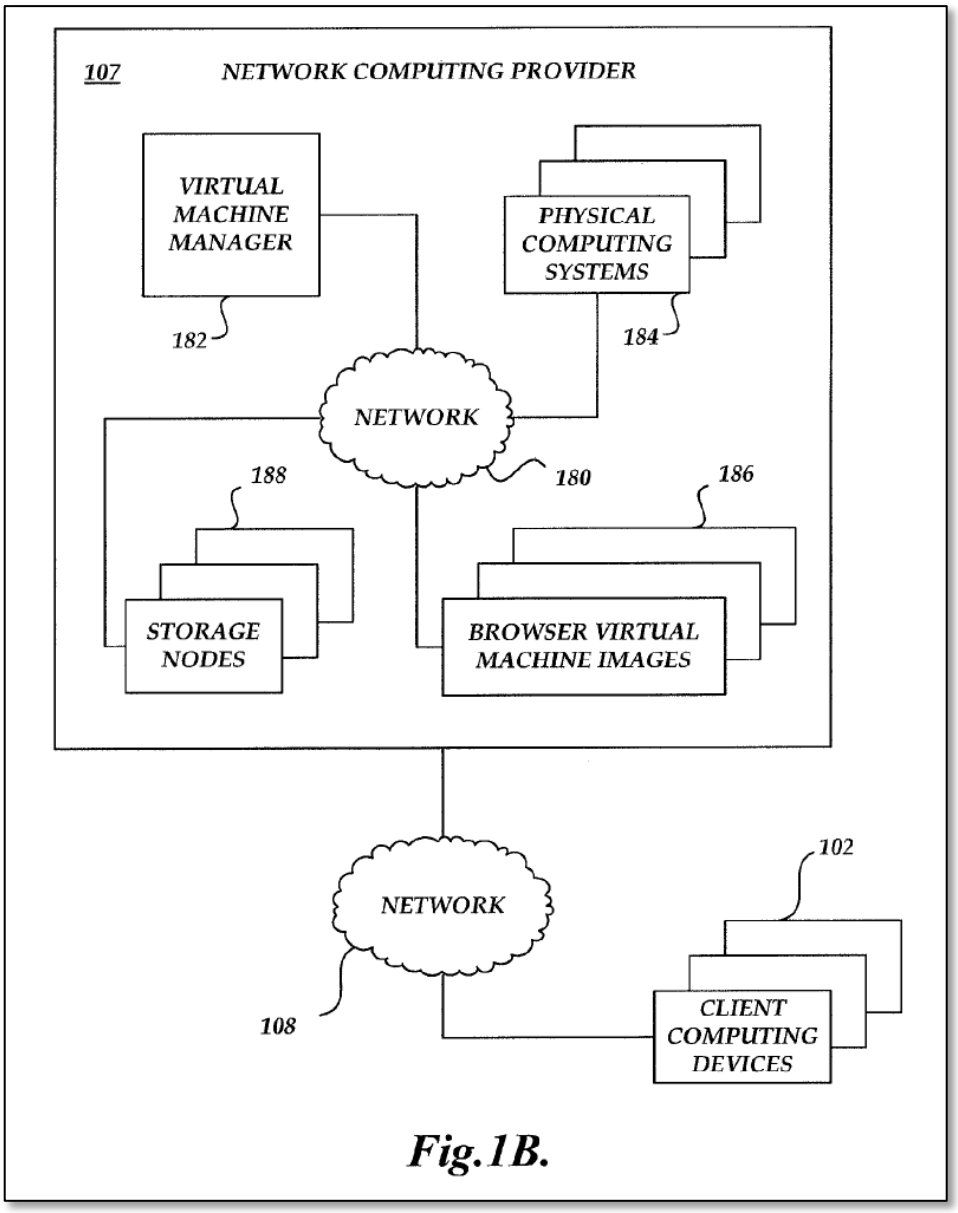


Fig. 1

Fages (EX1004), Fig. 1.



Hayden (EX1050), Fig. 1B.

220. Jai explains that its invention can operate on any computer system. Jai (EX1006), 13:3-7 ("The present invention may be used in a variety of computing devices. A 'computer device' is defined herein as any data processing device including microprocessors, conventional personal computers … The term is meant to be construed broadly."). Further, both Fages and Korycki depict standard

computer hardware such as memory, inputs, communication modules, processors, network connections, a server, among other common pieces of hardware. Fages (EX1004), Fig. 1; Korycki (EX1005), Figs. 1-2.

221. **Third**, Hayden and Jai teach implementing browser instance detector functionality, URL-comparison functionality, and its other elements as computer programming instructions stored in memory. *Compare* Hayden (EX1050), 38:32-55 ("Any process descriptions, elements, or blocks in the flow diagrams described herein and/or depicted in the attached FIGURES should be understood as potentially representing modules, segments, or portions of code which include one or more executable instructions for implementing specific logical functions or steps in the process.") *with* Jai (EX1006), 12:8-23 ("This redundancy checking feature may be user programmable."). A POSITA would have understood that a feature being "programmable" would have included modifying programming instructions, which are stored in memory. *See* §VI.A above.

222. **Fourth**, a POSITA would have understood that modifying Fages would thus have required only straightforward changes to programming instructions, which would have been well within the skillset of a POSITA. Modifying programming instructions would have been an activity that a POSITA would have been familiar with, whether by performing the programming operation (*e.g.*, writing code) or utilizing/adapting code written by others. For example, Fages already teaches a

process by which an administrator can load/define/customize URL lists to server 20. Fages (EX1004), [0085]. A POSITA would have known that changing of URL lists includes modifying a programming language code (C/C++) and executing commands and utilities (*e.g.*, in a shell script).

223. **Fifth**, a POSITA would have understood that Fages's native method for modifying server 20 would have been one method to modify Fages with Jai's and Hayden's teachings with a reasonable expectation of success. In this example, regardless of the author of the programming instructions, a POSITA would have been able to utilize a native process in Fages to load programming instructions to the server for execution. Thus, the Jai URL-comparison functionality and Hayden browser instance detection functionality, both of which may be embodied in programming instructions, would have been able to be successfully loaded and utilized in the modified Fages using a process that Fages already teaches.

224. **Sixth**, Fages does not specify that a particular computer operating system or environment is required for its invention. For example, Fages is not limited to a particular operating system like Windows or a Linux distribution. Similarly, neither Jai nor Hayden is limited to a particular computer operating system or environment. Thus, a POSITA would have understood that modifying Fages per Hayden and Jai would not have rendered Fages unsatisfactory for its

purposes. Indeed, modifying Fages in this way would have enhanced Fages's functionality with a reasonable expectation of success.

10. Claim 1(d)(i): "implementing a daemon outside the first browser instance that tracks active browser instances and launches a new browser in order to manage web access according to the data defining the first web content class and the second web content class,"

225. I note that limitations 1(d)(i)-1(d)(iv) relate to claim elements including a daemon, a native application, and a browser extension. The prior art combination as mapped in §VII.A.10 above relies on Fages's overall system (as modified by Jai and Hayden) and further implements a daemon, a native application, and a browser extension as taught by Korycki.

226. As an initial matter, it is my opinion that a daemon is not a particularly "special" or "unique" programming technique. The '721 Patent did not invent daemons nor use daemons in a way different than how daemons had already been used in the prior art. Daemons have been known and utilized for many years prior to the '721 Patent. Daemons do not inherently demand special hardware or programming techniques. Implementing a daemon would have been akin to implementing other software programs (*e.g.*, modifying programming instructions that are stored in memory), which would have been well-within the skillset of a POSITA. *See* §VI.E above (opinion regarding daemons).

a. Fages's Teachings

227. Fages teaches a first browser instance, and the Fages-Jai-Hayden combination teaches confirming active browser instances and URLs and launching a new browser instance to manage web access (*e.g.*, by utilizing "partitioning") according to data defining the first web content class and second web content class. *See* §§VII.A.6–VII.A.9 above, Claim limitations 1(a)-1(d) (opining regarding Jai's URL-comparison functionality and Hayden's browser instance confirming functionality to confirm that an instance is not already running a requested URL). The Fages-Jai-Hayden combination does not expressly teach utilizing a daemon to perform this functionality, but a POSITA would have been motivated to modify the Fages-Jai-Hayden combination to use a daemon, as taught by Korycki.

b. Korycki's Teachings

228. Korycki teaches performing computer security by checking whether a user is allowed to access certain content, such as a web application, which is "executed in a first web browser instance." Korycki (EX1005), 1:10-21, 1:66-2:31. Specifically, Korycki teaches a "**daemon**" 218 that accesses a list of approved partners for a service web application and determines "whether the domain of the partner associated with the partner web application 212 (which was indicated in the subscriber request message) is present in the ACL [Access Control List], i.e., whether the service provider has approved the partner web application 212 for

communication with the service provider web application 204." *Id.*, 8:41-9:8, 9:60-10:22.

229. Korycki teaches that "only desired partner web applications are allowed to communicate with the service provider web application," and "any other web application (other than those on the list of partners approved by the service provider) may be prevented from participating in communication with the service provider web application." *Id.*, 1:66-2:31. In one example, Korycki teaches "comparing a domain associated with the partner web application with the entries of the list of partners approved by the service provider" to determine "whether the partner web application is approved for communication." *Id.*, 3:9-13.

230. Korycki teaches that daemon 218 is utilized for performing background processes related to the device security described in §VII.A.10.b above. *Id.*, 7:21-33 ("The publish-subscribe daemon application 218 is a native part of the service provider client application and is always running in the background at the device 102."), 9:60-10:22, 6:66-7:6.

231. A POSITA would have understood that a daemon such as daemon 218 is a background process/program that performs its functions automatically without requiring user intervention. *See* §VI.E above. For example, Korycki's daemon 218 runs as a background process on a computer system. Korycki (EX1005), 7:21-33. Thus, a POSITA would have understood that Korycki's daemon 218 can execute the

tasks it was programmed to perform without requiring a user to specifically request that the functions be performed. Likewise, other daemons would have performed the tasks they are programmed to perform.

232. Korycki's annotated Fig. 2 below depicts daemon 218 communicating with browser plugin 208 ("native application") and pub-sub.js widget 206 ("browser extension"), which are both depicted as part of web browser instance 202 (referred to as Browser 1 below). As indicated in Fig. 2, daemon 218 and browser plugin 208 are technology "native to the device platform of the device 102 (*e.g.*, C++)." *Id.*, 7:21-33.

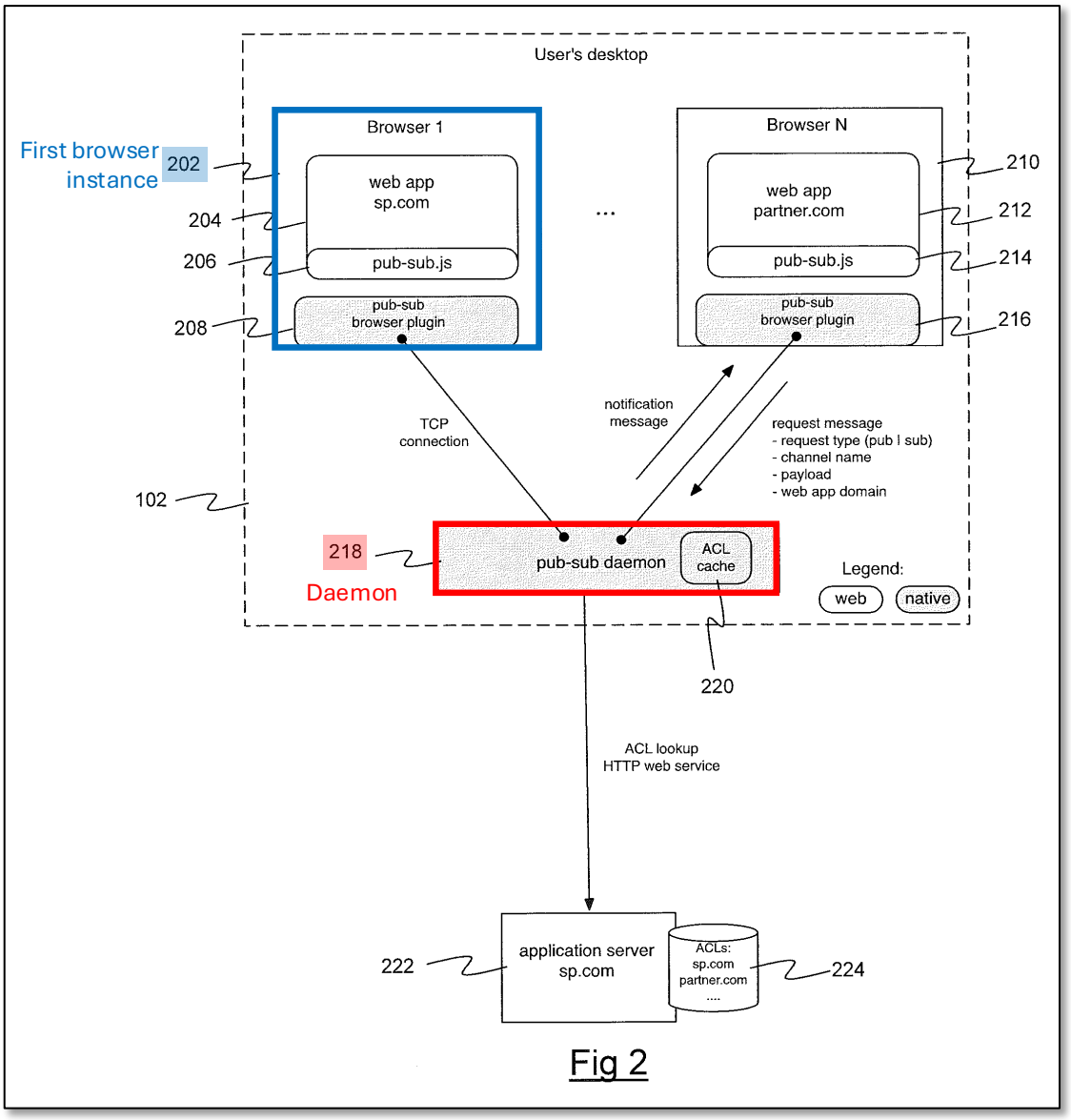


Fig 2

Id., Fig. 2 (annotated).

233. Korycki teaches that daemon 218 is outside browser 1, for example, and that "signals can be transmitted between the publish-subscribe daemon application 218 and the publish-subscribe browser plugin 208 and between the publish-subscribe daemon application 218 and the publish-subscribe browser plugin 216 using the publish-subscribe signaling scheme" which "may be achieved by

establishing respective Transport Control Protocol (TCP) connections." *Id.*, 7:34-49, Fig. 2.

234. **In sum**, the Fages-Korycki-Jai-Hayden combination teaches "implementing a daemon [Korycki's daemon 218 or another suitably programmed daemon] outside the first browser instance that tracks active browser instances [Hayden's browser instance confirmation/detection functionality and Jai's URL-comparison functionality that collectively tracks active browser instances] and launches a new browser in order to manage web access according to the data defining the first web content class and the second web content class."

c. Motivation to Combine Fages-Jai-Hayden with Korycki (Utilizing a Daemon was Advantageous and Well Known)

235. A POSITA would have been motivated to utilize Korycki's daemon to perform the Fages-Jai-Hayden's functionality of tracking active browser instances and URLs and launching a new browser to manage web access according to the data defining the first and second web content classes for at least the following seven reasons.

236. **First**, daemons were a well-known, commonly utilized piece of software/set of programming instructions implemented to perform background tasks that a user would not have needed to actively control. *See* §VI.E above. Fages already teaches automatically performing its functionality but does not expressly

teach the implementation details of what programming technique is utilized for performing its background tasks (*e.g.*, using a daemon).

237. **Second**, using a daemon would have been a logical programming option that a POSITA would have considered from the set of known programming options and that a POSITA would have been motivated to select. A POSITA would have understood and been motivated to modify Fages to implement its functionality using the well-known technique a daemon to advantageously, for example, monitor web browsers instances outside the web browsers because it is easier to monitor one process from another, rather than from inside the same process.

238. **Third**, monitoring one process externally using another (often called a "nanny" process in Unix), was not just easier but also more robust for two reasons. First, if monitoring takes place inside the monitored process, if the monitored process terminates abruptly (*e.g.*, due to a bug or attack), then the monitoring component will have been terminated, and thus could not alert an administrator of the failed process. Second, monitoring resources and usage is easier externally, using operating system facilities such as system call traces and resource usage calls (*e.g.*, rusage). Conceptually, this is akin to have an armed guard designated to protect a court house: it makes more sense to monitor everyone as they attempt to come into the building, rather than when they are already deep inside.

239. **Fourth**, a POSITA would have understood that daemons can be deployed in devices with a range of processing capabilities, including even in devices with limited resources such as embedded devices. Cardona (EX1009), [0013] ("Embedded devices tend to have limited resources, such as reduced memory and computing speed. Accordingly, the duplication of processes such as the network communication daemons can severely test the limited resources available in embedded devices. Reductions in the number of processes that are required to perform a specific task, such as monitoring network traffic, can result in smaller, less expensive devices that require fewer resources."). Thus, Cardona teaches that daemons are well-suited even for use within embedded devices, which often have limited resources.

240. **Fifth**, a POSITA would have been motivated to modify the Fages-Jai-Hayden combination further with Korycki to achieve Korycki's benefits. Using a daemon would have been one of a finite set of known programming techniques for accomplishing various tasks advantageously. For example, Korycki teaches utilizing a well-known computer programming element such as daemon 218 to achieve benefits such as higher performance, lower delay, and lower cost. Korycki (EX1005), 11:24-12:30. Each of these benefits individually would have motivated a POSITA to utilize a daemon, and the combination of multiple benefits would have further motivated a POSITA.

241. **Sixth**, modifying Fages with Korycki would have been use of a known technique that would have improved the similar Fages-Jai-Hayden system in the same way as Korycki. Implementing a daemon would have simply been using of a known technique to achieve Fages's functionality using known programming methods such as taught by Korycki.

242. **Seventh**, combining prior art elements such as Fages's computer system and server with well-known computer programming techniques (*e.g.*, daemon) according to known methods would have yielded the predictable result of a functional Fages-Jai-Hayden-Korycki computer system.

243. A POSITA would have had a reasonable expectation of success when modifying Fages-Jai-Hayden as per Korycki.

244. I first note that daemons were a well-known and well-understood programming technique before the priority date of the '721 Patent, such that a POSITA would have been familiar with daemons and would have known how to implement a daemon in a computer system. For example, Korycki teaches implementing its components including daemon 218 "in software for execution by the CPU 106 on the device 102" and specifically daemon 218 "may be provided by way of a computer program product embodied on a non-transient computer-readable medium which is configured so as when executed on the CPU 106 of the device 102 to perform the method steps described herein." Korycki (EX1005), 13:36-61. As

noted in §VII.A.9.e above, the Fages-Korycki-Jai-Hayden combination utilizes programming instructions stored in memory, thus modifying the Fages-Korycki-Jai-Hayden combination would have required only modifications to programming instructions, which a POSITA would have been capable of achieving.

245. Further, Fages, Jai, Hayden, and Korycki teach similar computer systems, such that Fages's hardware would not have needed to be modified to implement Korycki's modification. *See* §V above. Indeed, implementing a daemon such as Korycki's would not have required specialized hardware; a typical computer system such as taught by Fages, Korycki, Jai, and/or Hayden would have been sufficient to implement a daemon. Like in §VII.A.9.e above when modifying Fages as per Jai and Hayden, modifying the Fages-Jai-Hayden combination with Korycki would have utilized Fages's native server-modification process, where an administrator would have modified server 20 to utilize daemon 218 (or a similarly programmed daemon). In this example, a POSITA would have been able to load daemon programming instructions to server 20 to implement a daemon such as daemon 218 or another suitably programmed daemon to perform the tasks described herein.

246. Again, there is nothing inherently "special" or "difficult" or "unique" about utilizing a daemon, as a daemon is simply a type of background program/process used to perform particular tasks for which it was programmed.

This modification would have required only straightforward changes to programming instructions, which would have been well within the skillset of a POSITA. Indeed, I developed, maintained, and wrote daemons as part of my previous work and educational experience. *See* §I above, Educational Background and Qualifications.

247. While Korycki teaches implementing its functionality in a "client-side" setup, Korycki would have been compatible with Fages because Fages's setup involves a server executing browsers and communicating with web servers. Moreover, a user at a client device can remotely access the browser via a remote control program such as "NX protocol" from "NOMACHINE (www.nomachine.com)," "Remote Desktop Protocol (RDP) by Microsoft or Independent Computing Architecture (ICA) by Citrix," for example. Fages (EX1004), [0064], [0108].

248. In my opinion, Fages is not limited to these particular programs, but they are merely exemplary remote-control programs that Fages lists as suitable. Fages teaches that this remote-control setup allows for the system to function "as if the user were in front of the screen of the machine on which the Internet browser is executed. However, by means of this software program, the user can actually be physically at a great distance from this machine." *Id.*, [0009], [0004]-[0005], [0011], [0048], [0064]-[0071] ("The remote-controlled Internet browsers are the same as

those that could be directly executed on the user stations without the assistance of the modules 40 and 52."). Thus, a POSITA would have understood that the Fages-Jai-Hayden combination, when modified by Korycki, would have functioned similarly to how Korycki functions on a client computer that executes Internet browsers locally.

249. Indeed, a POSITA would have understood that Fages teaches that its remote-control program is essentially invisible to the user and that the browsers would have been operational whether executing locally on the user station or executing remotely at the server. Therefore, for the modifications proposed herein, Fages being implemented as "server-side" functionality and Korycki being implemented as a "client device" would have been essentially irrelevant to the proposed combination and would not have impeded the proposed modifications. Thus, modifying the Fages-Jai-Hayden combination with Korycki would have had a reasonable expectation of success.

11. Claim 1(d)(ii): "receiving the request via a browser extension associated with the first browser instance,"

a. Korycki's Browser Extension

250. Fages does not expressly teach a browser extension for its browsers. However, browser extensions were well known in the prior at. For example, Korycki teaches utilizing pub-sub.js 206 (also referred to as "widget 206"), which a POSITA would have understood functions as a "*browser extension*" of browser 202. Korycki

(EX1005), 6:40-65 ("The service provider web application 204 includes a widget 206 which is a JavaScript Application Programming Interface (API). The widget 206 allows the service provider web application 204 to communicate using the publish-subscribe signaling scheme. A publish-subscribe browser plugin 208 is implemented in the first web browser instance 202.").

251. This is consistent with the disclosures of the '721 Patent. '721 Patent (EX1001), 10:9-31 (describing a "JavaScript" browser extension for Firefox and Chrome). Indeed, Korycki's pub-sub.js 206 "widget 206" would have extended the functionality of browser 202 by providing communication with a native application (and ultimately to daemon 218). In general, Korycki's pub-sub.js 206 extends the browser's native communication functionality. As I discussed in §VI.B above, browser extensions were well-known and widely utilized software/programming instructions to extend a browser's functionality beyond that which the browser has natively. *See* §VI.B above.

252. I further note that Korycki teaches a plurality of browser instances and browser extensions. For example, similar to pub-sub.js 206 of browser 202, pub-sub.js 214 of browser 210 functions as a browser extension. Korycki (EX1005), 6:40-65. Korycki teaches pub-sub.js 206 communicates with pub-sub browser plugin 208 ("*native application*," *see* §VII.A.12 below, Claim 1(d)(iii)), which in

turn communicates with daemon 218 via a TCP connection, for example. Korycki (EX1005), 6:40-65, 8:41-9:8, Fig. 2.

253. Specifically, Korycki teaches a "widget 206 which is a JavaScript Application Programming Interface (API). The widget 206 allows the service provider web application 204 to communicate using the publish-subscribe signaling scheme." *Id.*, 6:40-65, 7:21-49 ("As shown in FIG. 2, the publish-subscribe daemon application 218 and the publish-subscribe browser plugins 208 and 216 are implemented using a native technology, that is, a technology that is native to the device platform of the device 102 (e.g. C++). The publish-subscribe daemon application 218 is a native part of the service provider client application and is always running in the background at the device 102 when the service provider client application has been downloaded to, and is implemented at, the device 102. In contrast, the web browser instances 202 and 210, the web applications 204 and 212, the widgets 206 and 214 and the service provider server 222 are implemented using web-based technology (e.g. in accordance with the HTML 5 standard). Signals can be transmitted between the publish-subscribe daemon application 218 and the publish-subscribe browser plugin 208 and between the publish-subscribe daemon application 218 and the publish-subscribe browser plugin 216 using the publish-subscribe signaling scheme. This may be achieved by establishing respective Transport Control Protocol (TCP) connections between the publish-subscribe

daemon application 218 and the publish-subscribe browser plugin 208 and between the publish-subscribe daemon application 218 and the publish-subscribe browser plugin 216. The publish-subscribe browser plugins 208 and 216 provide bridges between the respective web applications 204 and 212 and the publish-subscribe daemon application 218. The publish-subscribe daemon application 218 is also arranged to communicate with the service provider server 222 over the Internet 112.").

254. In this way, Fages's modified system "*receiv[es] the request* [to access web content] *via a browser extension* [pub-sub.js] *associated with the first browser instance*." Korycki's annotated Fig. 2 below depicts browser 202 including pub-sub.js 206:

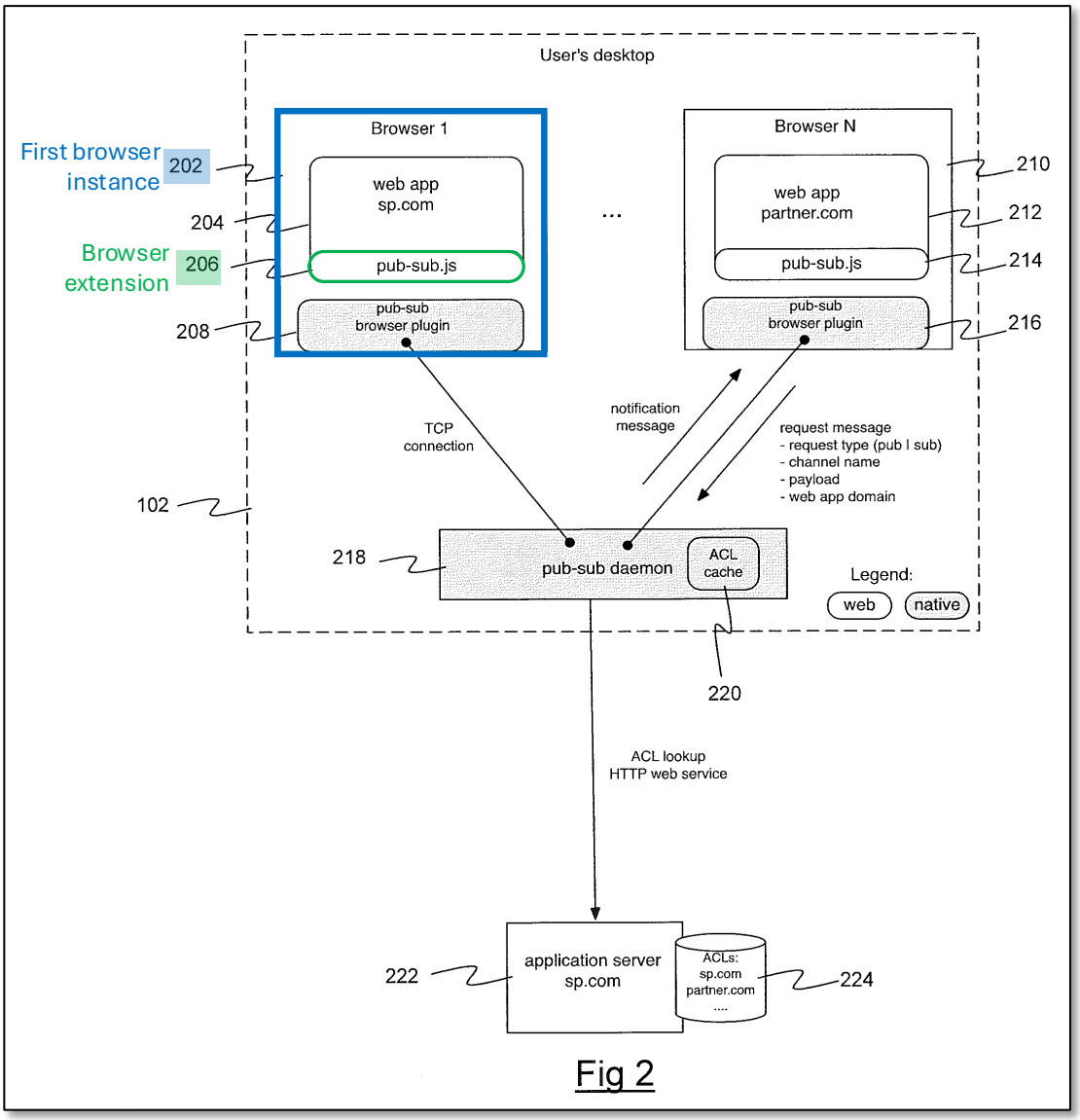


Fig 2

Id., Fig. 2 (annotated).

255. **In sum**, the Fages-Korycki-Jai-Hayden combination teaches "receiving the request via a browser extension [such as Korycki's browser extension pub-sub.js 206] associated with the first browser instance."

b. Motivation to Combine Fages-Jai-Hayden with Korycki (Browser Extension)

256. A POSITA would have been motivated to modify the Fages-Jai-Hayden combination with Korycki's teachings to include a browser extension such as pub-sub.js 206 in the Fages-Jai-Hayden system, for at least the following seven reasons.

257. **First**, as I opined in §VI.B above, adding a browser extension would have been a well-known technique to extend the functionality of a browser, such as by adding enhanced communication functionality (*e.g.*, to communicate with a native application). In this manner, a user would interact with the modified Fages's system, and the request to access a URL would be routed to the browser extension, on to the native application, and then to a daemon for approval or denial based on the URL and URL lists.

258. **Second**, similar to Korycki's daemon 218 discussed in §§VII.A.8 above, VII.A.9 above, and VII.A.10 above, the Fages-Jai-Hayden combination already teaches performing its browser security functionality but does not expressly teach the implementation details of what programming technique/component is utilized for extending a browser's functionality and providing communication with other processes. A POSITA would have been motivated to utilize a well-known programming technique such as including a browser extension in the Fages-Jai-Hayden-Korycki system to advantageously extend the functionality of browsers (*e.g.*, Fages's browsers 36-38).

259. **Third**, utilizing a browser extension in the Fages-Jai-Hayden combination as per Korycki would have provided Fages-Jai-Hayden with a well-known and widely utilized component for communicating with a native application and daemon (*e.g.*, through the native application) and would have extended the browser's functionality by providing a more versatile communication functionality. Browser extensions were a well-known, commonly utilized piece of software/set of programming instructions implemented to extend browser functionality and provide communication with other processes/components in the computer system (*e.g.*, native application). *See* §VI.B above.

260. **Fourth**, a POSITA would have understood and been motivated to modify Fages-Jai-Hayden to implement Korycki's functionality using the well-known technique of implementing a browser extension like pub-sub.js 206 to advantageously customize their browser(s) and improve functionality across web pages for tasks important to the user (*e.g.*, security). As I discussed in §VI.B above, browser extensions offered several such advantages. *See* §VI.B above (opining regarding Google Team (EX1017), 20).

261. **Fifth**, a POSITA would have been motivated to modify Fages-Jai-Hayden for the same reasons discussed in §VII.A.10 above (including applying similar motivation for including a browser extension in addition to a daemon). *See* §VII.A.10 above, Claim 1(d)(i). I note that the same or similar advantages as I

discussed in §VII.A.10 above would also apply herein regarding the browser extension.

262. **Sixth**, a POSITA would further have been motivated to modify Fages-Jai-Hayden with Korycki to achieve Korycki's benefits. For example, Korycki teaches enhanced security by utilizing a browser extension, native application, and daemon to implement Access Control List (ACL) functionality to determine if a user may access a particular web resource. Korycki (EX1005), 7:7-20 ("The database 224 includes a list of domains which are trusted by the service provider, and which is maintained and kept up-to-date by the service provider. If a domain of a partner on the service provider is trusted by the service provider then communication is allowed to proceed between the service provider web application 204 and the partner web application 212 at the device 102 … Indications of the trusted partners of the service provider are stored in the database 224 as an Access Control List (ACL) defining which partners are allowed for mash-up with the service provider web application 204 at the device 102."), 9:34-10:43, 11:24-49, 13:7-35 ("The publish request message may undergo a domain check corresponding to that performed for the subscribe request as described above. In particular, the publish-subscribe daemon application 218 may determine whether the partner web application 212 which sent the publish request message is allowed to communicate using the publish operation by checking that the domain of the partner web application 212 is included

in the ACL (stored at the database 224 or in the cache memory 220). Based on the determination, the publish-subscribe daemon application 218 then either allows or prevents communication from the partner web application 212 using the publish request."), 13:51-61 (describing an ACL list).

263. Modifying Fages-Jai-Hayden with Korycki would have amounted to using a known technique that would have improved the similar Fages-Jai-Hayden system in the same way as Korycki's system. In particular, using a browser extension was a well-known programming technique to enhance the functionality of a web browser.

264. **Seventh**, combining prior art elements such as Fages-Jai-Hayden's computer system and server with well-known computer programming techniques (*e.g.*, daemon, browser extension) according to known methods would have yielded the predictable result of a functional Fages-Jai-Hayden-Korycki computer system. As I discuss in next, modifying Fages-Jai-Hayden with Korycki would have resulted in a computer system including browsers with browser extensions added for improved functionality and security.

265. A POSITA would have had a reasonable expectation of success when modifying Fages-Jai-Hayden with Korycki. For example, Korycki teaches implementing its components including pub-sub.js 206 "in software for execution by the CPU 106 on the device 102" and specifically pub-sub.js 206 "may be provided

by way of a computer program product embodied on a non-transient computer-readable medium which is configured so as when executed on the CPU 106 of the device 102 to perform the method steps described herein." Korycki (EX1005), 13:36-61. Because the browser extension is implemented in computer programming instructions, a POSITA would have had a reasonable expectation of success to modify Fages according to Fages's own method for updating software/programming instructions at the server (*e.g.*, an administrator loads the programming instructions in one example).

266. Moreover, Fages, Jai, Hayden, and Korycki teach similar computer systems, such that Fages's hardware would not have needed to be modified to implement Korycki's modification. As discussed in §§VII.A.9 above, VII.A.10 above, and V above, Fages alone or in view of Korycki already teaches a processor, memory, web browsers, and network interface hardware/functionality, such that only Korycki's browser extension programming instructions would need to be added to the modified Fages system to achieve the benefits I discussed and ensure a reasonable expectation of success.

267. Similar to the discussion in §§VII.A.9 above and VII.A.10 above, when modifying Fages with Jai, Hayden, modifying Fages-Jai-Hayden with Korycki would have utilized Fages's native server modification process, where an administrator would have modified server 20 to utilize pub-sub.js 206. Fages

(EX1004), [0085]-[0086]. This modification would have required only straightforward changes to programming instructions, which would have been well-within the skillset of a POSITA. As I discussed in §I above, I had experience modifying programming instructions. *See* §I above, Educational Background and Qualifications.

12. Claim 1(d)(iii): "forwarding the request from the browser extension to the daemon using a native application,"

268. As explained in §§VII.A.10–VII.A.11 above, Fages-Jai-Hayden-Korycki teaches using a daemon such as daemon 218 and a browser extension such as pub-sub.js 206. *See* §§VII.A.10–VII.A.11 above, Claim limitations 1(d)(i)-1(d)(ii).

a. *Korycki's* Teachings

269. Fages-Jai-Hayden does not expressly teach utilizing a "native application" in its system to communicate between a browser extension and daemon ("*forwarding the request from the browser extension to the daemon using a native application*"). Korycki teaches such a native application that communicates between pub-sub.js 206 ("*browser extension*") and a *daemon* (*e.g.*, daemon 218).

270. Korycki's pub-sub.js 206 ("*browser extension*") is a JavaScript Application Programming Interface (API) implemented using "web-based technology (*e.g.*, in accordance with the HTML 5 standard)." Korycki (EX1005), 6:40-55, 7:21-33. Korycki teaches that daemon 218 and publish-subscribe browser

plugins 208 and 216 are "implemented using a native technology, that is, a technology that is native to the device platform of the device 102 (e.g., C++)." *Id.* Thus, a POSITA would have understood that pub-sub browser plugin (*e.g.*, 208, 216) was a *native application* at least because the browser plugins are "implemented using a native technology" such as in accordance with the HTML 5 standard.

271. Furthermore, Korycki teaches that pub-sub browser plugin 208 performs the same functionality as the claimed *native application*. Korycki teaches that pub-sub.js 206 ("*browser extension*") communicates with pub-sub browser plugin 208 ("*native application*") because "widget 206 allows the service provider web application 204 to communicate using the publish-subscribe signaling scheme," and pub-sub browser plugin 208 communicates with daemon 218, for example, "by establishing respective Transport Control Protocol (TCP) connections," (where plugin 208 and daemon 218 are both implemented using a C-based language, *e.g.*, C++) ("*forwarding the request from the browser extension to the daemon using a native application*"). *Id.*, 6:40-65, 7:34-48, 8:44-9:8 (teaching similar communication between widget 214 and browser plugin 216 where "**widget** 214 **delegates the subscribe request to** the **native publish-subscribe browser plugin** 216.").

272. Moreover, a POSITA would have understood that pub-sub.js 206 assists web application 204 to communicate with browser plugin 208 and daemon

218, such that "publish-subscribe browser plugins 208 and 216 provide bridges between the respective web applications 204 and 212 and the publish-subscribe daemon application 218," and where pub-sub.js 206 "allows the service provider web application 204 to communicate." *Id.*, 7:44-46, 6:51-55.

273. I note that this communication path in Korycki (*e.g.*, from browser extension to native application to daemon) is the same as described and claimed in the '721 Patent. *See* '721 Patent (EX1001), Claim 1, 10:16-20 ("Since the extensions cannot directly communicate with the daemon, the system has a native application written in C that is responsible for forwarding messages between the daemon and the extension."). Korycki's annotated Fig. 2 below depicts a browser 202 which includes pub-sub browser plugin 208 ("*native application*") and daemon 218, where both plugin 208 and daemon 218 are shaded to indicate they are "native" technology:

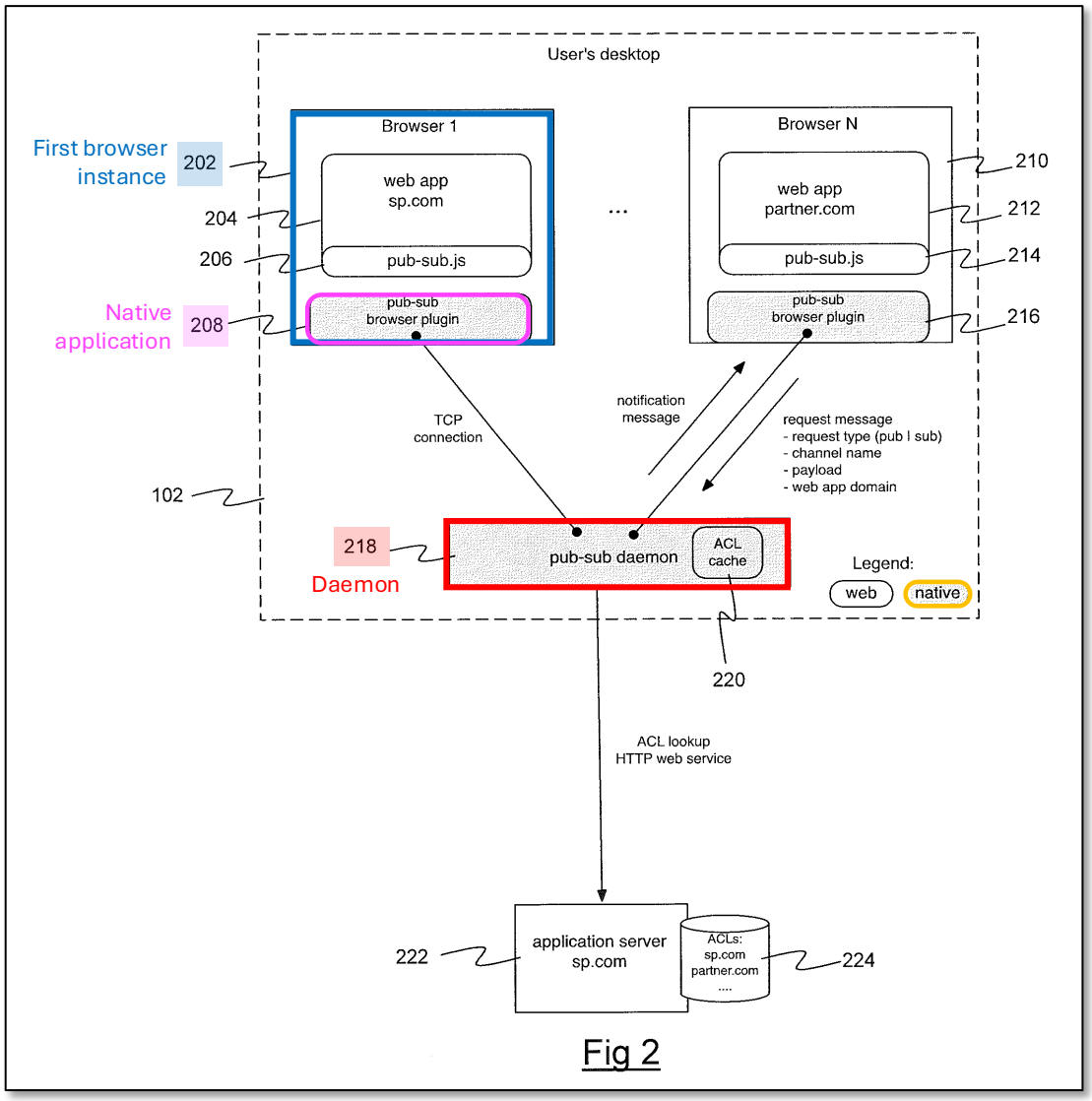


Fig 2

Korycki (EX1005), Fig. 2.

274. Korycki describes this further. *Id.*, 1:22-32 ("includes installed (native) components deployed at the device in the form of browser plugins and other installed software that augments the pure web components of the service provider client application."), 1:66-2:19, 7:21-33 ("the publish-subscribe browser plugins 208 and 216 are implemented using a native technology, that is, a technology that is native to the device platform of the device 102 (e.g., C++)."), 8:54-9:8, 11:53-12:12.

275. I also note that communication would also work in reverse from daemon 218 to plugin 208 as indicated by the two-way line in Fig. 2. A POSITA would also have understood that two-way communication would have existed because daemon 218 must communicate instructions with plugin 208.

276. **In sum**, the Fages-Korycki-Jai-Hayden combination teaches "forwarding the request from the browser extension [Korycki's pub-sub.js 206] to the daemon [Korycki's daemon 218] using a native application [Korycki's browser plugin 208, 216]."

b. Motivation to Combine Fages-Jai-Hayden with Korycki (Native Application Provides Communication)

277. A POSITA would have been motivated to modify Fages-Jai-Hayden with Korycki's teachings to include a native application such as browser plugin 208 in the Fages-Jai-Hayden-Korycki system, for at least the following six reasons. In this manner, a user would interact with the modified Fages's system, and the request to access a URL would be routed to the browser extension, on to the native application, and then to a daemon for approval or denial based on the URL and URL lists.

278. **First**, similar to the daemon 218 and pub-sub.js 206 discussed in §§VII.A.9 above, VII.A.10 above, and VII.A.11 above, Fages-Jai-Hayden already teaches performing its browser security functionality but does not expressly teach

the implementation details of what programming technique/component is utilized for communicating between a browser extension and daemon. A POSITA would have been motivated to utilize a well-known programming technique such as including a browser plugin ("*native application*") in the Fages-Jai-Hayden-Korycki system to advantageously provide communication between the browsers and a daemon such as daemon 218.

279. **Second**, utilizing a browser plugin ("*native application*") in Fages-Jai-Hayden as per Korycki would have provided Fages-Jai-Hayden with a well-known and widely utilized component for communicating with a browser extension and daemon and would have extended the browser's functionality by providing said communication functionality between components based on different technologies (*e.g.*, browser extension based on HTML 5 and daemon 218 based on a C language). Native applications were a well-known, commonly utilized piece of software/set of programming instructions implemented to extend browser functionality and provide communication with other processes/components in the computer system (*e.g.*, native application). *See* §VI.G above. A POSITA would have understood and been motivated to modify Fages-Jai-Hayden with Korycki to implement its functionality using the well-known technique of implementing a native application like browser plugin 208 which is advantageously browser-agnostic and can access features of the

native device. Furthermore, the native application can interface with available APIs within the operating system. *See* §VI.G above and §VI.H above.

280. **Third**, a POSITA would have been motivated to modify Fages-Jai-Hayden for the same reasons discussed in §§VII.A.10–VII.A.11 above (including applying similar motivation for including a browser extension in addition to a daemon). *See* §§VII.A.10–VII.A.11 above, Claim limitations 1(d)(i)-1(d)(ii). I note that the same or similar advantages as I discussed in §§VII.A.10–VII.A.11 above would also apply herein regarding the browser plugin ("native application").

281. **Fourth**, a POSITA would further have been motivated to modify Fages-Jai-Hayden with Korycki to achieve Korycki's benefits of using a native application. For example, Korycki teaches enhanced security by utilizing a browser extension, browser plugin ("native application"), and daemon to implement Access Control List (ACL) functionality to determine if a user may access a particular web resource. Korycki (EX1005), 7:7-20, 9:34-10:43, 11:24-49, 13:7-35, 13:51-61.

282. **Fifth**, modifying Fages-Jai-Hayden with Korycki would have amounted to using a known technique that would have improved the similar Fages-Jai-Hayden system in the same way as Korycki. In particular, using a browser plugin was a well-known programming technique to enhance the functionality of a web browser and to provide communication between various components in a computer system.

283. **Sixth**, combining prior art elements such as Fages-Jai-Hayden's computer system and server with well-known computer programming techniques (*e.g.*, daemon, browser extension, native application) according to known methods would have yielded the predictable result of a functional Fages-Jai-Hayden-Korycki computer system. As I discuss in §§VII.A.9 above, VII.A.10 above, and VII.A.11 above, modifying Fages-Jai-Hayden-Korycki would have resulted in a computer system including browsers with browser extensions and browser plugins added for improved functionality and security.

284. A POSITA would have had a reasonable expectation of success when modifying Fages-Jai-Hayden with Korycki. For example, Korycki teaches implementing its components including browser plugin 208 "in software for execution by the CPU 106 on the device 102" and specifically browser plugin 208 "may be provided by way of a computer program product embodied on a non-transient computer-readable medium which is configured so as when executed on the CPU 106 of the device 102 to perform the method steps described herein." Korycki (EX1005), 13:36-61. Because the browser plugin is implemented in computer programming instructions, a POSITA would have had a reasonable expectation of success to modify Fages according to Fages's own method for updating software/programming instructions at the server (*e.g.*, an administrator loads the programming instructions in one example).

285. Lastly, Fages, Jai, Hayden, and Korycki teach similar computer systems, such that Fages's hardware would not have needed to be modified to implement Korycki's modification. As discussed in §§VII.A.9 above, VII.A.10 above, and VII.A.11 above, Fages alone or in view of Korycki already teaches a processor, memory, web browsers, and network interface hardware/functionality, such that only Korycki's browser plugin programming instructions would need to be added to the modified Fages system to achieve the benefits I discussed and ensure a reasonable expectation of success. Similar to the discussion in §§VII.A.9 above, VII.A.10 above, and VII.A.11 above when modifying Fages with Jai and Hayden, modifying Fages-Jai-Hayden with Korycki would have utilized Fages's native server modification process, where an administrator would have modified server 20 to utilize browser plugin 208. Fages (EX1004), [0085]-[0086]. This modification would have required only straightforward modifications to programming instructions, which would have been well-within the skillset of a POSITA. As I discussed in §I above, I had experience modifying programming instructions. *See* §I above, Educational Background and Qualifications.

13. Claim 1(d)(iv): "confirming, via the daemon that the web content is not associated with the first web content class and that another browser instance is not already launched and available to access the web content; and"

286. Fages as modified by Jai and Hayden teaches determining which browser instances and URLs are active and/or inactive by comparing requested

URLs against open URLs and browser instances ("*confirming … another browser instance is not already launched and available to access the web content*"). *See* §VII.A.9 above, Claim 1(d). Fages as modified by Korycki teaches utilizing a daemon to perform background processing tasks. *See* §§VII.A.10–VII.A.12 above, Claim limitations 1(d)(i)-1(d)(iii). Fages teaches determining whether a requested URL is associated with a first web content class. *See* §VII.A.6 above, Claim 1(a). Fages does not expressly discuss utilizing a daemon for this process. Fages as modified by Korycki teaches utilizing a daemon to perform this task. *See* Claim limitations 1(d)(i)-1(d)(iii) (§§VII.A.10–VII.A.12 above).

287. Korycki teaches performing security functions with daemon 218 including accessing an Access Control List (ACL) to determine if a browser can access a particular website. Korycki (EX1005), 7:34-49, 9:34-11:5 (including teaching that daemon 218 performs security tasks such as "accessing the list (i.e., the ACL) of approved partners" to determine if a browser may access said partner (*e.g.*, website/web application)), 13:7-35, 6:16-21 ("web applications may be web pages downloaded over the network").

288. In this proposed combination, Fages's security process would have been modified to include a daemon to perform such background security processes. This modification would have been a natural and straightforward modification to the Fages system, to take advantage of the well-known capabilities and benefits of

daemons. In particular, these tasks are advantageously performed in the background, as taught by Korycki, such that when Fages is modified to utilize a daemon, Fages's security tasks also would have been performed using the daemon.

289. Thus, in the Fages-Jai-Hayden-Korycki combination, a daemon would have been utilized to perform background processes such as confirming that web content is associated with a first or second web content class. This use of a known technique would have improved the similar Fages and Korycki systems in the same way. Combining prior art elements such as a daemon and a computer system according to known methods would have yielded predictable results, such as a functional computer system utilizing a daemon to perform background processing tasks including related to security. A POSITA would have been motivated to modify Fages-Jai-Hayden-Korycki and would have had a reasonable expectation of success for the same reasons discussed in §§VII.A.10–VII.A.12 above. *See* §§VII.A.10–VII.A.12 above, Claim limitations 1(d)(i)-1(d)(iii).

290. **In sum**, the Fages-Korycki-Jai-Hayden combination teaches "confirming, via the daemon that the web content is not associated with the first web content class [Fages's non-sensitive content class] and that another browser instance is not already launched and available to access the web content; and."

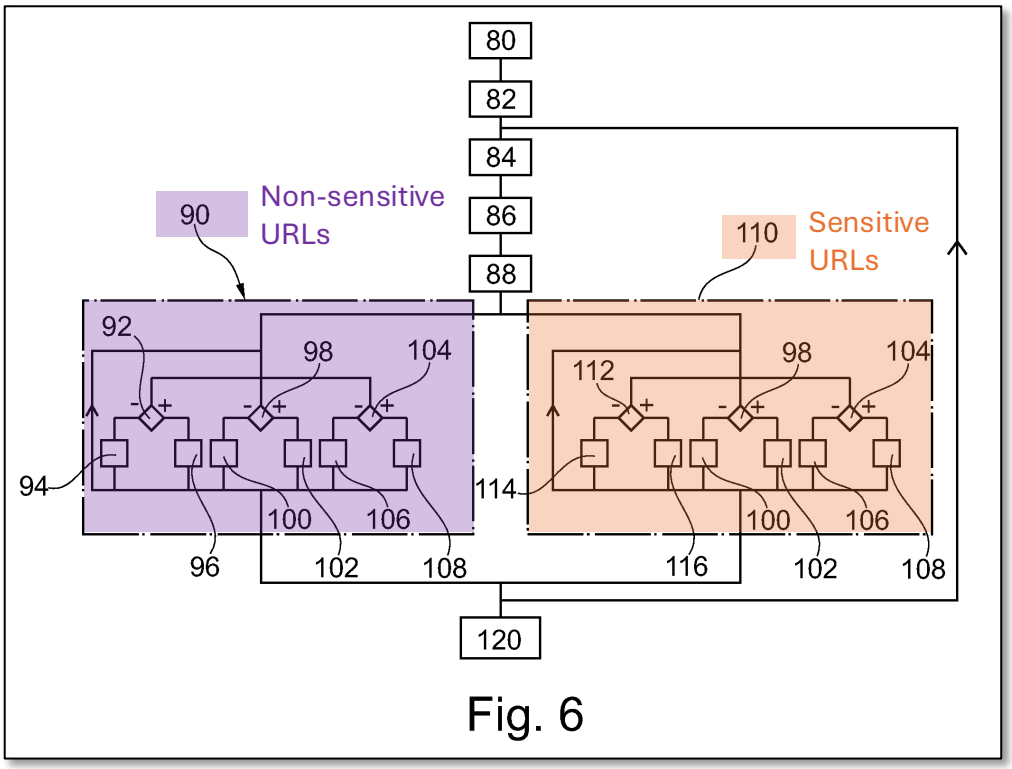
14. Claim 1(e): "launch a new browser for accessing the web content associated with the second web content class to provide

access to the URL of the second plurality of URLs via the new browser."

291. Fages teaches "parallel execution of several Internet browsers in execution environments that are independent of one another" such that if a "URL belongs to a pre-recorded list of URLs of sensitive Internet sites to which connection from the first execution environment is not authorized," then Fages launches a new browser in parallel in a "second execution environment" "to connect to the sensitive Internet site identified by this URL[.]" Fages (EX1004), [0044]-[0046], [0098]-[0099], [0104], [0109]-[0110] ("When several Internet sessions are executed simultaneously, they are displayed in distinct windows on the screen 58" or "in different tabs of a same window."), *id.* ("server 20 automatically launches the execution of another Internet browser in another execution environment"); *see* §VII.A.6 above, Claim 1(a).

292. A POSITA would have understood that if a user attempts to access non-sensitive/non-secure sites using browser 36 via session S_4 in execution environment 32, for example, then another browser instance (*e.g.*, 37, 38) would be utilized for a secure session *e.g.*, S_1 to access a URL in list L_{11} in execution environment 33 or 34. Fages (EX1004), Fig. 3. *See* §§VII.A.6–VII.A.13 above, Claims limitations 1(a)-1(d). In my opinion, this comports with Fages's express teachings of "partitioning" non-sensitive/non-secure and sensitive/secure browser instances. Fages (EX1004), [0104], [0109]-[0110].

293. A POSITA would have understood that by utilizing separate browser instances for sensitive and non-sensitive content, security is enhanced by preventing sensitive content from being accessed in the same browser instance as non-sensitive content. In my opinion, this is in line with Fages's express goals and teachings. *Id.*, [0104], [0109]-[0110]. Fages depicts separate blocks for non-sensitive URLs 90 and sensitive URLs 110 in annotated Fig. 6 below:



Id., Fig. 6 (annotated).

294. In one example, Fages's session S_4 would be used to access a non-sensitive URL in browser instance 36 of execution environment 32, and a second browser instance to access a sensitive URL would be opened in execution environment 33 and browser instance 37 using session S_1. I note that this is merely

one example, and a POSITA would have understood that other browser instances and browser execution environments may have been used in a similar manner the sensitive and non-sensitive web content. Regardless, a POSITA would have understood that in the modified Fages system, a new browser instance is launched to handle sensitive content when no such browser instance for handling sensitive content is currently open. *See* §VII.A.7 above, Claim 1(b).

295. **In sum**, the Fages-Korycki-Jai-Hayden combination teaches "launch a new browser for accessing the web content associated with the second web content class to provide access to the URL of the second plurality of URLs via the new browser."

15. Claim 2: "The system of claim 1, wherein the new browser instance is launched automatically for display, and the web content is associated with a web server."

296. Fages teaches that "rather than blocking a URL with which it is not possible to get connected through the Internet browser executed in the current execution environment, the server 20 **automatically launches the execution of another Internet browser** in another execution environment from which the connection to this URL is possible." Fages (EX1004), [0109]-[0110]. *See also id*., [0024]-[0025], [0035], [0038]-[0039], [0092], [0100], [0109]-[0110]. Indeed, a POSITA would have understood that Fages would have automatically launched a new browser instance when appropriate, including because Fages teaches

automatically launching the execution of another Internet browser instance and because Fages does not teach requiring a user to perform the browser launching operation manually.

297. Fages also teaches the web content is associated with a web server. For example, Fages teaches Internet servers 8 and 10, where in one example, Internet server 8 "is an Internet server that enables online handling of bank accounts." *Id*., [0056], [0088] ("Thus, a sensitive site such as an Internet site enabling the management of a user's bank accounts will be accessible only by means of an Internet browser whose configuration defined in the list C_i is appropriate to this use."), [0091]-[0093], [0098]-[0100], [0008].

298. Thus, a POSITA would have understood that Fages's web content is associated with a web server such as servers 8 and/or 10. I note that while Fages teaches only a single web server (*e.g.*, server 8) for sensitive content (*e.g.*, online banking), a POSITA would have understood that one or more servers to handle sensitive web content would have been utilized. Additionally, a POSITA would have understood that a server such as server 8 that "enables online handling of bank accounts" indicates that the web content (*e.g.*, online banking) is associated with a web server (*e.g.*, server 8, the server that handles online banking).

299. **In sum**, the Fages-Korycki-Jai-Hayden combination teaches "[t]he system of claim 1, wherein the new browser instance is launched automatically for

display [Fages's browser], and the web content is associated with a web server [Fages's server 8]."

VIII. OPINION: GROUND 2

A. Obviousness over Fages, Korycki, Jai, Hayden, and Russello

300. Claims 3-5 would have been obvious over Fages, Korycki, Jai, Hayden, and Russello.

1. Overview of Russello

301. U.S. Patent Application Publication No. 2014/0137184 to Russello *et al.* ("Russello") was filed October 10, 2013 and claims priority to foreign applications dated November 13, 2012 and July 25, 2013 (EX1007, cover).

302. Russello teaches utilizing a process tracer call ("ptrace") to intercept "all or at least a set of system calls executed by a target process." Russello (EX1007), [0026]. In Russello, "the tracing system call may be a process trace system call (ptrace)." *Id.*, [0026], [0036], [0047]-[0048], [0092], [0127], [0196], [0203] ("The term 'monitoring' as used in this specification and claims is intended to mean, in the context of a master-target process relationship, where the master intercepts, controls, modifies and/or traces system calls made by the target process to the kernel, and may also include or be referred to technically as 'interposing' and/or 'interleaving."), [0250].

303. Russello also teaches implementing a "process monitor" in conjunction with ptrace to control system calls and execute security policies related to ptrace

operations. *Id.*, [0026]-[0031], [0036], [0040]-[0049], [0052]-[0056], [0064], [0092], [0196], [0203]-[0204], [0249]-[0252], [0256], [0258], [0293]-[0294], [0296]. In general, Russello teaches the process monitor is the "master" process that is associated with a "target process," and system calls are traced such that system calls executed by the target process are intercepted and reported to the master process (process monitor). *Id.*

2. Claim 3: "The system of claim 2, wherein the processor is operable to execute an API call configured to intercept a system call to the web server, the system call to the web server intended to provide access of the URL to the first browser instance."

a. '721 Patent's API

304. The '721 Patent describes intercepting system calls using a ptrace API. '721 Patent (EX1001), 7:16-31 ("This API allows one program (tracer) to be notified and take control of every system call made by another program (browser)."), 8:62-65, 9:22-26. The '721 Patent claims that the API performs calls and controlling system calls. '721 Patent (EX1001), Claims 3-4.

b. Russello's API Call

305. Russello teaches utilizing "a process tracer call (ptrace)" to intercept "all or at least a set of system calls executed by a target process." Russello (EX1007), [0026], [0036], [0047]-[0048], [0092]. For example, Russello teaches that its ptrace intercepts system calls "if a parent process launches a new child process" where "any system calls invoked by the new child process are intercepted

and redirected to a monitoring entity associated with the parent process to implement a security action." *Id.*, [0127]; *see also id*. [0196] ("The term 'attached' or 'attaching' as used in this specification and claims is intended to mean, unless the context suggests otherwise, creating a relationship between two processes in the Linux layer whereby one process is configured to monitor and control the other process in parent-child or master-target relationship, and by way of example includes attachment by way of a process trace system call, e.g., ptrace."), [0203] ("The term 'monitoring' as used in this specification and claims is intended to mean, in the context of a master-target process relationship, where the master intercepts, controls, modifies and/or traces system calls made by the target process to the kernel, and may also include or be referred to technically as 'interposing' and/or 'interleaving."), [0250].

306. Additionally, a POSITA would have understood that in Russello, ptrace is utilized including in web browser processes, for example, in a Linux-based Google Chrome environment, which executes web-based applications. *Id.*, [0406]-[0407]. Russello teaches monitoring processes in the operating system and that a monitoring entity 812 (which indicates using ptrace, as I discussed in the preceding paragraph) intercepts system calls to enforce security policies. *Id.*, [0414]-[0415]. Thus, a POSITA would have understood that Russello teaches using its ptrace functionality in a web browser-related scenario.

307. Russello also teaches implementing a "process monitor" in conjunction with ptrace to control system calls and execute security policies related to ptrace operations. *Id.*, [0026]-[0031], [0036] ("Each process monitor may be configured to enforce security policies based on parameters extracted from the detected system calls."), [0040]-[0049] ("a process monitor attached by the main monitor process to each new process created by the mother process and each process monitor being configured to monitor the system calls made by its attached process to the kernel."), [0052]-[0056], [0064], [0092], [0196], [0203]-[0204], [0249]-[0252], [0256], [0258], [0293]-[0294], [0296]. In general, Russello teaches that the process monitor is the "master" process that is associated with a "target process," and system calls are traced such that system calls executed by the target process are intercepted and reported to the master process (process monitor). *Id*.

308. Russello's ptrace functionality is an *API* that controls system calls and enables Russello's process monitor to control system calls. A POSITA would have understood that the process monitor is a tracer (*see* §VIII.A.6 below, Claim 5) and the ptrace functionality is an *API* including because the '721 Patent's description of a tracer program and an API overlaps with Russello's teachings, and because the '721 Patent describes ptrace (a known API in the prior art, for example, used in Linux systems) as an *API*. '721 Patent (EX1001), 7:16-31 ("the system includes a

ptrace API available in Linux"), 8:62-9:26 ("Linux ptrace API" and describing "tracer").

309. In my opinion, the ptrace functionality described in the '721 Patent is the same as the ptrace functionality taught in Russello, at least because the ptrace system call is standardized in POSIX-based operating systems. Thus, Russello's process monitor is a tracer using a ptrace *API* that performs a *call configured to intercept a system call*. In combination with Fages's modified system, Russello's ptrace performs Fages's interception functionality such that it "*intercept[s] ... the system call to the web server intended to provide access of the URL to the first browser instance*." This understanding of Russello is consistent with what I opined in §VIII.A.2.b above when discussing Russello's process monitor and ptrace functionality including in the context of web browsers.

310. The process monitoring functionality of Russello using a ptrace API (including controlling system calls) is further mapped for Claim 4(b) (§VIII.A.5 below) and the process monitoring functionality is mapped for the tracer in Claim 5 (§VIII.A.6 below).

311. Russello teaches implementing its process monitoring and ptrace functionality in a web browser embodiment. Russello (EX1007), [0406]-[0415]. In this example, Russello teaches utilizing a "monitoring entity 812" to perform ptrace functionality. *Id.,* [0203] ("The term 'monitoring' as used in this specification and

claims is intended to mean, in the context of a master-target process relationship, where the master intercepts, controls, modifies, and/or traces system calls made by the target process to the kernel, and may also include or be referred to technically as 'interposing' and/or 'interleaving.'"), [0406]-[0415].

312. **In sum**, the Fages-Korycki-Jai-Hayden-Russello combination teaches "The system of claim 2, wherein the processor is operable to execute an API call [Russello's ptrace functionality] configured to intercept a system call [Fages's system using ptrace] to the web server, the system call to the web server intended to provide access of the URL to the first browser instance."

c. Motivation to Combine Fages-Korycki-Jai-Hayden with Russello (ptrace and API)

313. A POSITA would have been motivated to modify Fages-Korycki-Jai-Hayden with Russello to achieve Russello's security benefits, for at least the following five reasons.

314. **First**, intercepting API calls allows for the system to perform security processing before allowing an action to be completed (*e.g.*, a user selecting a URL to browse to). In the modified Fages system, this modification would have provided a mechanism to pause browsing operations while security processes are performed to enhance Fages's security.

315. **Second**, a POSITA would have understood that Russello teaches improving system security using its teachings, such as using a process trace (ptrace)

to intercept system calls and using process monitoring to enforce security policies (among other functionality), including in a web browsing example. Russello (EX1007), [0015]-[0016], [0036], [0047]-[0048], [0406]-[0415] (teaching utilizing the security system for Chromium, including "code downloaded visiting a web page, executing a web application, and other code that it will be rendered and executed by Chromium processes."). As I discussed in §VI.F above, utilizing ptrace was well-known in the prior art. *See* §VI.F above.

316. **Third**, this use of a known technique would have improved the similar Fages-Korycki-Jai-Hayden system in the same way as Russello's computer system by implementing process monitoring and ptrace functionality in the Fages-Korycki-Jai-Hayden combination. Fages, Korycki, Jai, Hayden, and Russello all teach similar computer systems, such that modifying the Fages-Korycki-Jai-Hayden combination with Russello would have achieved Russello's security benefits and would have provided low-level implementation details not expressly described in Fages. *See* §V above where I opine regarding compatibility with Linux, for example.

317. **Fourth**, combining prior art elements such as Russello's ptrace and process monitor with the Fages-Korycki-Jai-Hayden computer system's elements according to known methods would have yielded the predictable result of increased system security. Indeed, Russello's ptrace and process monitor would have allowed

for system calls to be intercepted to perform additional security processing before allowing a user to browse a URL.

318. **Fifth**, while Fages already teaches a security process for checking URL lists, Fages does not expressly describe how system calls are handled. Russello provides these low-level implementation details that would have benefited the Fages-Korycki-Jai-Hayden combination. Thus, this modification would have informed a POSITA how to achieve additional security benefits by providing low-level implementation details not expressly taught in Fages.

319. A POSITA would have had a reasonable expectation of success when modifying Fages-Korycki-Jai-Hayden with Russello. I note that Russello teaches the well-known ptrace and process monitoring functionality that were present in prior art computer systems, such as Linux. Russello (EX1007), [0015]-[0016], [0026]-[0031], [0035]-[0036], [0196]-[0197], [0203]-[0204], [0215].

320. Fages-Korycki-Jai-Hayden does not expressly teach a specific operating system is required, such that a POSITA would have understood that using techniques for Linux would also have functioned in the modified Fages-Korycki-Jai-Hayden-Russello system. Indeed, a POSITA would have understood that an operating system such as Linux would have been compatible with the Fages-Korycki-Jai-Hayden combination, including because none of Fages, Korycki, Jai or

Hayden disparage, discredit, or otherwise teach away from using a Linux implementation.

321. Furthermore, Russello's security benefits including for web browsing are achieved in a Linux environment, leading to a reasonable expectation of success. Modifying Fages-Korycki-Jai-Hayden with Russello would have required only straightforward changes to programming instructions, which would have been well-within the skillset of a POSITA.

322. To the extent it is argued that modifying Fages-Korycki-Jai-Hayden with Russello would have required utilizing a specific operating system environment such as Unix or Linux, a POSITA would have been motivated to modify Fages-Korycki-Jai-Hayden-Russello to utilize Linux to achieve Russello's security benefits and because Linux was a well-known, well-understood, and commonly utilized operating system choice in the prior art. Russello (EX1007), [0015]-[0016], [0026]-[0031], [0035]-[0036], [0196]-[0197], [0203]-[0204], [0215]; *see* §VI.I above.

323. A POSITA would have had a reasonable expectation of success implementing Fages-Korycki-Jai-Hayden-Russello in Linux because Russello is already implemented in Linux, and because modifying Fages-Korycki-Jai-Hayden to operate in Linux would have merely required straightforward modifications to programming instructions, an otherwise standardized (*e.g.*, POSIX) operating

system. *See* my detailed discussion in §V above, including opining that Fages was compatible with Linux. Fages (EX1004), [0064] (describing "NX protocol"), where I note that NX protocol was originally designed for Linux operating systems. *See* §V above.

324. I note that functions such as ptrace were already available in Linux, such as described by the '721 Patent and taught in Russello. Thus, implementing the Fages-Korycki-Jai-Hayden combination in Linux would have been merely a matter of modifying programming instructions to implement the combination in Linux.

3. Claim 4(Pre): "The system of claim 3, further comprising:"

325. *See* §VIII.A.2 above, Claim 3.

4. Claim 4(a): "wherein the processor is operable to pause operations associated with the first browser instance upon interception of the system call; and"

326. As an initial matter, I note that any time a process invokes a system call, it is often done by performing a user-level interrupt/trap instruction (sometimes called "int 80h"). When the user level process initiates the interrupt associated with a system call, the main CPU suspends the user process, then the OS kernel takes over in privileged mode, to execute the system call on behalf of the now-suspended process. Only when the kernel is done executing the system call, does it resume executing the user process. Therefore, by the very nature of how Unix/Linux

operating systems behave, a user process (*e.g.*, web browser) is paused each time it executes a system call. I have personally taught these steps many times. *See* §I above.

a. Russello's Pausing Operations

327. Russello teaches using process trace (ptrace) and a processor monitor to pause operations associated with a system call. Russello (EX1007), [0026] ("the kernel is configured to report to the master process system calls made by the target process both prior to and after the execution of the system call, and in some embodiments, before the kernel actually resumes the target process execution."), [0250] (teaching that the kernel utilizes process monitoring and ptrace functionality to "**suspend the target process**" and "await a security action from the process monitor" to resume the target process), [0343]-[0346] ("The child process 375 also raises a SIGSTOP 380 to be **stopped** until the parent process 370 allows it to resume its execution.").

328. Thus, operations would have been paused before resuming target process execution. Indeed, Russello's system call interception process would have required pausing operations by not immediately performing a requested operation (*e.g.*, a user selecting a URL to browse to). Thus, when modifying Fages-Korycki-Jai-Hayden with Russello, the processor would have been operable to pause

operations associated with the first browser instance upon interception of the system call by utilizing Russello's process-monitoring functionality and ptrace API.

329. Finally, I note that Russello's pausing is similar to the '721 Patent's described pause process. '721 Patent (EX1001), 7:17-24. Furthermore, the modified Fages's processor would have been operable to resume operations (*e.g.*, resuming a target process execution) as discussed in §VIII.A.4.b below. Russello (EX1007), [0343]-[0346] ("The child process 375 also raises a SIGSTOP 380 to be stopped until the parent process 370 allows it to resume its execution.").

330. **In sum**, the Fages-Korycki-Jai-Hayden-Russello combination teaches "wherein the processor is operable to pause operations [Russello's pausing operations] associated with the first browser instance upon interception of the system call [Russello intercepting system calls]; and."

b. Motivation to Combine Fages-Korycki-Jai-Hayden with Russello (Pausing Operation Benefits)

331. A POSITA would have been motivated to modify the Fages-Korycki-Jai-Hayden combination with Russello for the reasons discussed in §VIII.A.2 above, and which apply herein. *See* §VIII.A.2 above, Claim 3.

332. Furthermore, a POSITA would have been motivated to modify the Fages-Korycki-Jai-Hayden combination to achieve Russello's security benefits by pausing browser operations upon interception of the system call, to allow sufficient

time to perform security operations and to prevent processing from occurring until the security procedures have been completed.

333. I note Claim 4 does not require any specific length of time for a pause to occur, such that the modified Fages would have included a processor that is operable to pause operations associated with the first browser instance upon interception of the system call for some unspecified length of time as needed to perform its security process before resuming operations (such as taught by Russello).

334. Modifying Fages-Korycki-Jai-Hayden with Russello would have had a reasonable expectation of success for the reasons discussed in §VIII.A.2 above. *See* §VIII.A.2 above, Claim 3. Specifically, modifying Fages-Korycki-Jai-Hayden would have required only straightforward modifications to programming instructions, which would have been well-within the skillset of a POSITA.

5. Claim 4(b): "wherein the API controls system calls made by other programs and other browser instances."

335. Fages-Korycki-Jai-Hayden does not expressly discuss the API controls system calls made by other programs and other browser instances. Specifically, Korycki teaches an API widget 206 for each browsing instance rather than a single API. Russello, however, teaches the API controls system calls made by other programs and other browser instances.

a. Russello's API Controls System Calls

336. Russello teaches that its ptrace API controls system calls for all programs/processes set as master/target when invoked by a process monitor, and all programs/processes set as master/target would have included other programs and other browser instances. As discussed in §VIII.A.2 above and §VIII.A.4 above, Russello teaches examples such as usings its process monitoring and ptrace functionality in the context of web browsers. *See* §VIII.A.2 above, Claim 3, and §VIII.A.4 above, Claim 4(a).

337. Furthermore, a POSITA would have understood that Russello's process monitoring and ptrace functionality would have been applicable to any other master and target processes established within the system. Indeed, a POSITA would have understood that Russello does not limit its master-target/parent-child monitoring and ptrace functionality only to specific applications or processes, but rather that this functionality would have extended to any desired master-target/parent-child processes. Russello (EX1007), [0026], [0036], [0047]-[0048], [0092] ("any system calls invoked by the new child process are intercepted and directed to the embedded process monitor of the parent process to implement a security action."), [0196], [0203]-[0204], [0258], [0267]-[0273] ("As the process monitor 72 is attached to the new process 74, it can intercept and control all the system calls that this process executes."), [0406]-[0415] ("each monitoring entity retrieves and enforces security policies specific to the application corresponding to its process.").

338. Therefore, a POSITA would have understood that any other program or other browser instance set as a master/target can utilize Russello's process monitoring and ptrace functionality; such a process would thus have benefited from Russello's improved security functionality by controlling system calls to said other programs and other browser instances in the same way as described in §VIII.A.4 above for the first browser instance. Therefore, in the context of the Fages-Korycki-Jai-Hayden-Russello combination, Russello's teachings of process monitoring and ptrace functionality for controlling API calls would have extended to all other programs and other browsing instances.

339. **In sum**, the Fages-Korycki-Jai-Hayden-Russello combination teaches "wherein the API controls system calls [Russello's API controlling system calls] made by other programs and other browser instances [Russello intercepting system calls from every program and would have included every browser instance]."

b. Motivation to Combine Fages-Korycki-Jai-Hayden with Russello (API Controls System Calls from Other Programs and Browser Instances)

340. A POSITA would have been motivated to modify Fages-Korycki-Jai-Hayden with Russello for the same reasons discussed in §VIII.A.2 above and §VIII.A.4 above to achieve Russello's improved computer security and benefits associated with such security. *See* §VIII.A.2 above, Claim 3, and §VIII.A.4 above, Claim 4(a).

341. In addition, a POSITA would have been motivated to apply Russello's teachings to all other programs and other browsing instances to achieve its security benefits throughout the entire system. This would have been a use of Russello's known technique that would have benefited the similar system of the Fages-Korycki-Jai-Hayden combination in the same way as Russello's system.

342. This modification would have been a natural, advantageous, and straightforward extension of the modification already discussed in §VIII.A.2 above and §VIII.A.4 above. *See* §VIII.A.2 above, Claim 3, and §VIII.A.4 above, Claim 4(a). Russello (EX1007), [0237]-[0238] ("The security system monitors all system calls and based on the relevant policy specified for a process they are either allowed or blocked."), [0259] ("the security system is configured to monitor all applications and corresponding processes").

343. A POSITA would have had a reasonable expectation of success because Russello teaches applying its functionality to a first browser instance and utilizing its process monitoring and ptrace functionality for other programs. Russello does not teach limiting its functionality to a single browser instance, such that a POSITA would have understood that Russello's process monitoring and ptrace functionality would have been operable on a plurality of other programs and browsing instances.

344. Lastly, modifying Fages-Korycki-Jai-Hayden with Russello would have required only straightforward modifications to programming instructions,

which would have been well-within the skillset of a POSITA. *See* §VIII.A.2 above, Claim 3, and §VIII.A.4 above, Claim 4(a). Additionally, Russello that teaches its security system monitors all system calls and allows or blocks each call based on the relevant policy (Russello (EX1007), [0237]-[0238], [0259]); this would have indicated to a POSITA that Russello's teachings would have applied to all the programs and browsing instances in the modified Fages's computer system with a reasonable expectation of success.

6. Claim 5: "The system of claim 1, wherein the processor is operable to implement a tracer to determine whether or not a requested network object is allowed to be accessed by the first browser instance."

345. As an initial matter, I note that a system such as ptrace, designed to trace all system calls, certainly includes tracing networking related system calls (*e.g.*, for opening a network socket, transmitting and receiving packets, and more).

a. '721 Patent's Tracer

346. The '721 Patent describes a tracer such as a program enabled by a ptrace API to take control of system calls of another program. '721 Patent (EX1001), 7:16-31 ("To intercept system calls, the system includes a ptrace API available in Linux. This API allows one program (tracer) to be notified and take control of every system call made by another program (browser)."), 8:62-65 ("Using Linux ptrace API, the system may trace every system call made by a child process to keep track of the application's access to system resources including network and file system."), 9:22-

26 (“The ptrace API allows one to trace a process and all its child processes and threads.”)

b. Russello’s Tracer (Process Monitor Using ptrace)

347. Fages-Korycki-Jai-Hayden does not expressly discuss implementing a tracer. Russello, however, teaches implementing a tracer program, such as a process monitor, enabled by “ptrace” that can, for example, track system calls made by a target process. Russello (EX1007), [0026] (“intercept all or at least a set of system calls executed by the target process and report those to the master process prior to the execution of such system calls.”), [0196] (“a process trace system call, e.g., **ptrace**, or any other mechanism that creates a link or attachment between processes which enables one process to control one or more aspects or functions of the other process.”), [0203]-[0204], [0250] (“When a process monitor invokes ptrace to attach to a target process, each time the target process invokes a system call the gateway will suspend the target process.”), [0258]. *See* §§VIII.A.2–VIII.A.5 above, Claims 3-4. Also, as I opined in §VIII.A.6.a above, Russello teaches that its process monitor and ptrace functionality can track all system calls for all processes in the system, which is also in line with the ’721 Patent’s description of a Linux ptrace. *See* §VIII.A.6.a above.

348. In the Fages-Korycki-Jai-Hayden combination, Russello’s ptrace would have allowed Fages’s system to utilize a tracer to track system calls and pause

processes to supplement Fages's native security functionality used to determine whether or not a requested network object is allowed to be accessed by a first browser instance. *See* §§VIII.A.2–VIII.A.5 above, Claims 3-4.

349. **In sum**, the Fages-Korycki-Jai-Hayden-Russello combination teaches "The system of claim 1, wherein the processor is operable to implement a tracer [Russello's process monitor using ptrace] to determine whether or not a requested network object is allowed to be accessed by the first browser instance."

c. Motivation to Combine Fages-Korycki-Jai-Hayden with Russello (Tracer Tracking System Calls)

350. A POSITA would have been motivated to modify Fages-Korycki-Jai-Hayden with Russello to implement a tracer such as a process monitor enabled by ptrace to track system calls, for example, from a first browser instance/session. Russello teaches tracking all system calls, which a POSITA would have understood to include system calls from a first browser instance. Fages already teaches checking URL lists to determine if a browser instance can access sensitive URLs. *See* §§VII.A.5–VII.A.14 above, Claim 1. A POSITA would have understood that Fages does not expressly teach utilizing a tracer to track system calls, which Russello teaches.

351. Specifically, Russello provides low-level implementation details for how to utilize a tracer to track system calls. Modifying the Fages-Korycki-Jai-Hayden combination with Russello would have amounted to using a known

technique to implement a well-known tracer such as a process monitor enabled by ptrace to track system calls and enhance Fages's security. This use of a known technique would have improved the Fages-Korycki-Jai-Hayden combination's similar computer system in the same way as Russello's similar computing system. This combination of prior art elements according to known methods would have yielded the predictable result of an improved Fages computer system, including because Russello provides low-level implementation details that Fages does not expressly teach. Combining these prior art elements would have resulted in an advantageous and improved computer system.

352. A POSITA would have had a reasonable expectation of success modifying the Fages-Korycki-Jai-Hayden combination because Fages already teaches a similar computing system and utilizing security such as URL lists. *See* §§VII.A.5–VII.A.14 above, Claim 1. However, Fages lacks implementation details for how to implement a process monitor and ptrace. Implementing Russello's process monitor tracer would have provided a well-known prior art solution for implementing additional security and would have enhanced the Fages-Korycki-Jai-Hayden combination in the same way as Russello.

353. As I opined in §§VIII.A.2–VIII.A.5 above, modifying the Fages-Korycki-Jai-Hayden combination with Russello would have achieved a reasonable expectation of success. *See* §§VIII.A.2–VIII.A.5 above, Claims 3-4. Furthermore,

modifying Fages-Korycki-Jai-Hayden with Russello would have required only straightforward modifications to programming instructions, which would have been well-within the skillset of a POSITA. *See* §§VIII.A.2–VIII.A.5 above, Claims 3-4.

IX. OPINION: GROUND 3

A. Obviousness over Fages, Korycki, Jai-Hayden, O'Niell, and Cardona

354. Claims 6-8, 12-13, and 16-18 would have been obvious over Fages, Korycki, Jai, Hayden, O'Niell, and Cardona.

1. Overview of O'Niell

355. U.S. Patent No. 8,424,017 to O'Niell *et al.* ("O'Niell") was filed on April 30, 2010; its Application Publication No. US 2011/0271087 was published November 3, 2011; and it was issued on April 16, 2013 (EX1008, cover).

356. O'Niell teaches a computer system utilizing daemon 235 and that daemon 235 utilizes Unix domain sockets for communication and is capable of utilizing other sockets such as Linux domain sockets. O'Niell (EX1008), 5:28-51 ("FIG. 2 is a block diagram representation of operating system 200 that a designer or other entity configures with a process ID embedding handler 205 and an embedded socket pathname checker 207 according to the disclosed methodology. Referring to both FIG. 1 and FIG. 2, IHS 100 stores operating system 200 in system memory 120 as operating system 200″. Operating system 200 includes user space 210 and kernel space 215. A fence 220 denotes a conceptual boundary between user

space 210 and kernel space 215. User space 210 may include one or more applications 225 that execute from user space 210. Applications 225 include an application 230 that initializes, spawns or otherwise launches a daemon process 235."), 5:62-6:21 ("In parallel with creation of a respective Unix domain socket for the particular daemon process 235, the process ID embedding handler 205 accesses or gets the process ID that corresponds to the particular daemon process 235, as per block 315. In one embodiment, the command for accessing or getting the process ID of the particular process is "getpid()". Obtaining the process ID in this manner occurs in user space 210.").

357. O'Niell teaches a daemon implemented as a C program and also that the daemon utilizes Unix domain sockets for communication. O'Niell teaches utilizing computer program code for carrying out operations and wherein the computer program code "may be written in any combination of one or more programming languages, including … Java, Smalltalk, C++ or the like and conventional procedural programming languages, such as the 'C' programming language or similar programming languages." *Id*., 4:26-41.

2. Overview of Cardona

358. U.S. Patent Application Publication No. 2003/0061257 to Cardona ("Cardona") was filed on September 25, 2001 and published on March 27, 2003 (EX1009, cover).

359. Cardona teaches that daemons are "agent programs that continuously operate, for example, on a server, and provide resources to client systems on the network. These background processes can handle a variety of tasks, including monitoring the network for incoming data." Cardona (EX1009), [0003], [0006]-[0009], Figs. 1-2.

360. Specifically, Cardona teaches that its daemon is a multithreaded daemon to handle multiple threads or tasks concurrently. *Id*., [0036] ("The exemplary daemon according to the invention is multithreading, that is, it spawns multiple threads or tasks that exist concurrently."), [0037] ("a single multithreaded daemon can service multiple connections."), [0053] ("The resulting daemon is referred to as a multithreaded universal daemon (MUD), because it is capable of simultaneously monitoring many connections, up to the number that was specified as a parameter in the create function."), [0055].

3. Claim 6: The system of claim 1, wherein the daemon comprises a multi-threaded C program that uses Unix domain sockets to communicate with the browser extension and other extensions.

361. Fages-Korycki-Jai-Hayden does not expressly discuss that the daemon comprises a multi-threaded C program that uses Unix domain sockets. Korycki teaches implementing the daemon in the C++ programming language, but not expressly the "C" programming language. Korycki (EX1005), 7:21-33. A POSITA

would have known that the C++ language is a superset of the C language, and that any C code can be compiled and run by a C++ compiler suite.

362. To the extent it is argued that a daemon implemented using the C++ programming language is not sufficient to meet the claimed *C program*, O'Niell teaches a daemon implemented as a C program. O'Niell further teaches a daemon that uses Unix domain sockets for communication. As discussed in §IX.A.3.b below, Cardona further teaches a C program daemon that is multi-threaded.

a. O'Niell's C program Daemon Using Unix Domain Sockets

363. O'Niell teaches a computer system that utilizes daemon 235, that daemon 235 utilizes Unix domain sockets for communication, and that is capable of utilizing other sockets such as Linux domain sockets. O'Niell (EX1008), 5:28-54 ("Network stack 240 includes a domain socket handler 245 that creates a socket, such as a **Unix domain socket or Linux domain socket**, in response to a request from application 230 or daemon process 235."), 5:62-6:30 ("process ID embedding handler 205 embeds the process name for the particular daemon process 235 in the socket pathname for the **Unix domain socket** that corresponds to the particular daemon process.").

364. O'Niell teaches utilizing computer program code for carrying out operations and wherein the computer program code "may be written in any combination of one or more programming languages, including … Java, Smalltalk,

C++ or the like and conventional procedural programming languages, such as the 'C' programming language or similar programming languages." *Id*., 4:26-41.

365. Therefore, O'Niell's daemon would have been programmed in the C programming language in one embodiment. Indeed, O'Niell provides an express teaching, suggestion, or motivation for utilizing the C programming language as one of the suitable options for programming a daemon.

366. Furthermore, O'Niell's C programming language daemon would have utilized Unix domain sockets in at least one embodiment, including the embodiment where the daemon is a C program. I note that this would have been in line with O'Niell's teachings of programming the daemon in the C programming language and the daemon utilizing Unix domain sockets for communication.

b. Cardona's Multithreaded Daemon

367. Cardona teaches implementing a multithreaded daemon in a computer system. Cardona (EX1009), Abstract ("A system and method for a multithreaded universal daemon (MUD) is described."). Cardona teaches that daemons are "agent programs that continuously operate, for example, on a server, and provide resources to client systems on the network. These background processes can handle a variety of tasks, including monitoring the network for incoming data." *Id*., [0003], [0006]-[0009], Figs. 1-2.

368. Specifically, Cardona teaches that its daemon is a multithreaded daemon to handle multiple threads or tasks concurrently. *Id*., [0036] ("**The exemplary daemon according to the invention is multithreading**, that is, **it spawns multiple threads or tasks that exist concurrently.**"), [0037] ("a single **multithreaded daemon** can service multiple connections."), [0053] ("The resulting daemon is referred to as a multithreaded universal daemon (MUD), because it is capable of simultaneously monitoring many connections, up to the number that was specified as a parameter in the create function, and the same library functions are used to create and edit daemons that work with any type of data, using any protocol."), [0055], Fig. 3.

369. **In sum**, the Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination teaches "[t]he system of claim 1, wherein the daemon comprises a multi-threaded [Cardona's multi-threaded daemon] C program [O'Niell's C program daemon] that uses Unix domain sockets [O'Niell's Unix domain socket for a daemon] to communicate with the browser extension and other extensions."

c. Motivation to Combine Fages-Korycki-Jai-Hayden with O'Niell and Cardona (C Programming and Unix Domain Sockets)

d. O'Niell Motivation

370. A POSITA would have been motivated to modify Fages-Korycki-Jai-Hayden with O'Niell to advantageously utilize a daemon written in the C

programming language ("*C program*") and where the daemon uses Unix domain sockets, for at least the following four reasons.

371. **First**, it is my opinion that the C programming language was well known and widely utilized in the prior art and would have offered widespread compatibility for a daemon when the daemon is written as a C program. Moreover, I note that implementing the daemon as a C program would have provided a standard programming language that combines features of both high-level and low-level languages and is a structured programming language that allows for complex programs to be broken into simpler programs (*i.e.*, functions). *See* §§VI.E above and VI.J above.

372. **Second**, additionally, C is a general-purpose programming language that can work efficiently in various environments including enterprise and home use. The Bell System Technical Journal (EX1036), 10 ("The UNIX operating system, the C programming language, and the many tools and techniques developed in this environment are finding extensive use within the Bell System and at universities, government laboratories, and other commercial installations."); *see* §VI.J above. Modifying Fages-Korycki-Jai-Hayden with O'Niell would have amounted to using a known technique that would have improved the similar Fages-Korycki-Jai-Hayden daemon and computer system in the same way as O'Niell by capturing the benefits of using the C programming language such as those I outlined in just above.

373. **Third**, regarding using Unix domain sockets, a POSITA would have been motivated to modify Fages-Korycki-Jai-Hayden to utilize Unix domain sockets for daemon communication to capture the advantages of Unix domain sockets. For example, using Unix domain sockets would have allowed the Fages-Korycki-Jai-Hayden daemon to use local files to send and receive data more efficiently instead of having to rely on network interfaces. Lewandowski (EX1039), 6 ("A socket is a software abstraction that is used to create a channel-like interface between processes."); Sinha (EX1033), 7-8 ("Socket is an endpoint of communication … The 'type' of socket determines the method and mode of data transfer in the socket.").

374. **Fourth**, a POSITA would have understood that programs that run on the same server can communicate with each other using Unix domain sockets to advantageously bypass network based sockets and protocols. Oracle – UNIX Domain Sockets (EX1040), 1 ("UNIX domain sockets communicate only between processes on a single host. Sockets in the UNIX domain are not considered part of the network protocols because they can only be used to communicate between processes on a single host."). This use of a known technique would have improved the similar Fages-Korycki-Jai-Hayden computer system and daemon in the same way as O'Niell's computer system and daemon by taking advantage of Unix domain sockets for communication.

375. This modification would have had a reasonable expectation of success for several reasons. The C programming language was a well-known and widely utilized programming language, and these modifications would have required only straightforward changes to programming instructions, which would have been well-within the skillset of a POSITA. Furthermore, Fages-Korycki-Jai-Hayden does not provide express requirements for a particular programming language to be used, which would have led a POSITA to choose a widely utilized and powerful language such as C for programming a daemon.

376. Indeed, Korycki discloses implementing the daemon "using a native technology, that is, a technology that is native to the device platform of the device 102 (e.g., C++)." Korycki (EX1005), 7:21-25. In a scenario where the computer of the Fages-Korycki-Jai-Hayden combination utilizes the C programming language, this technology would have been used to implement the daemon using a native technology, which would have been in line with Korycki's teachings as well. Also, programming a daemon in languages such as C was well known in the prior art, with O'Niell providing an express teaching of such, further leading a POSITA to a reasonable expectation of success.

377. Lastly, utilizing Unix domain sockets was a well-known and widely utilized technique which was compatible with the C programming language. *See* §VI.J above. Thus, when modifying Fages-Korycki-Jai-Hayden with O'Niell, the

computer system and daemon would have been able to utilize Unix domain sockets for communication (*e.g.*, to/from the daemon) without having to rely on networking protocols, although such native functionality in Korycki would have been maintained as optional. This modification would have required only straightforward changes to programming instructions, which would have been well-within the skillset of a POSITA.

378. **In sum**, Fages-Korycki-Jai-Hayden-O'Niell-Cardona teaches "the system of claim 1, wherein the daemon [Fages-Korycki-Jai-Hayden-O'Niell-Cardona daemon] comprises a multi-threaded [Cardona's program] C program [O'Niell using C] that uses Unix domain sockets [O'Niell using Unix domain sockets] to communicate with the browser extension and other extensions."

e. Cardona Motivation

379. A POSITA would have been motivated to further modify Fages-Korycki-Jai-Hayden-O'Niell with Cardona's teachings of implementing a daemon as a multithreaded program, for at least the following three reasons.

380. **First**, Cardona teaches the benefits of implementing a daemon as a multithreaded program including to be able to handle multiple threads or tasks concurrently, which is in line with the purpose of utilizing a multithreaded program. Cardona (EX1009), [0036]-[0037] ("The exemplary daemon according to the

invention is multithreading, that is, it spawns multiple threads or tasks that exist concurrently.").

381. **Second**, a POSITA would have been motivated to modify the Fages-Korycki-Jai-Hayden-O'Niell combination to allow a daemon to handle multiple threads and/or tasks at the same time. Implementing multithreading in this manner would have enhanced the efficiency and functionality of the daemon. This use of a known technique of implementing a daemon as multithreaded would have improved the similar Fages-Korycki-Jai-Hayden-O'Niell daemon in the same way as Cardona's daemon.

382. **Third**, in my opinion, Cardona's express teaching, suggestion, and/or motivation to utilize a multi-threaded daemon would have motivated a POSITA to modify the daemon in the Fages-Korycki-Jai-Hayden-O'Niell combination to be multithreaded to capture the benefits of a multithreaded daemon.

383. A POSITA would have had a reasonable expectation of success when modifying the Fages-Korycki-Jai-Hayden-O'Niell combination's daemon to be multithreaded. I note that multithreading may be achieved by utilizing multithreading programming techniques, which would have been within the skillset of a POSITA to implement.

384. Additionally, Korycki already teaches that its daemon 218 is connected to multiple browsers, but Korycki does not expressly teach that its daemon is

multithreaded. However, a POSITA would have understood that Korycki teaches or at least reasonably suggests to a POSITA that a daemon may be multithreaded. A POSITA would have understood that modifying a daemon that is designed to handle multiple tasks to be multithreaded would have allowed for the daemon to handle multiple tasks concurrently, and hence more efficiently. A POSITA would have found such multithreading to be advantageous to the modified Fages's daemon and system in general.

385. Thus, modifying Fages-Korycki-Jai-Hayden-O'Niell with Cardona would not have rendered the daemon (*e.g.*, Korycki's daemon 218) inoperable for its intended function nor would this modification have changed the daemon's principle of operation; rather, this modification would have provided additional beneficial functionality for handling multiple tasks/threads concurrently and efficiently, as a POSITA would have expected from utilizing a multithreaded daemon.

386. Lastly, a POSITA would have understood that Cardona's daemon is compatible with a server, such as Fages's server. Cardona (EX1009), [0012], [0015]. Therefore, in my opinion, the modification would have been functional when implemented in a server such as Fages's server in the Fages-Korycki-Jai-Hayden-O'Niell combination. Modifying the Fages-Korycki-Jai-Hayden-O'Niell

combination would have required only straightforward changes to programming instructions, which would have been well-within the skillset of a POSITA.

4. Claim 7(Pre): "A method for facilitating different categories of content to be displayed in different independent browser instances, the method comprising:"

387. To the extent that the preamble is limiting, Fages teaches a method for facilitating different categories of content (*e.g.*, sensitive, non-sensitive) to be displayed in different independent browser instances. *See* Claim 1 (§§VII.A.5–VII.A.14).

5. Claim 7(a): "utilizing a processor in communication with a non-transitory tangible storage medium storing instructions that are executed by the processor to perform operations comprising:"

388. Fages alone or in view of Korycki teaches a processor and non-transitory tangible storage medium storing programming instructions executed by a processor. *See* Claim 1(Pre) (§VII.A.5 above).

6. Claim 7(a)(i): "accessing data defining rules for delegating web access associated with a first content class to a first browser type and delegating web access associated with a second content class to a second browser type;"

389. *See* Claim 1(a) (§VII.A.6 above). A POSITA would have understood that the data defining which browser instances may access which web content are "rules," as claimed. Indeed, this data would have been understood to be security policies/rules for how particular browsers must behave within the system. Thus,

Fages's URL lists and rules/policies regarding the use of the URL lists would have been understood to be "*data defining rules*."

7. Claim 7(a)(ii): "identifying a request to access an IP address by a first browser instance, the first browser instance being of the first browser type;"

390. Fages teaches lists defining URLs. *See* Claim 1(b) (§VII.A.7 above). A POSITA would have understood that a URL would have specified or been related to a particular IP address, such that identifying a request for a particular URL would have also identified a request to access an IP address related to said URL. *See* §VI.D above (explaining how a URL is converted into a corresponding IP address via a Domain Name System (DNS) query). Therefore, Fages teaches identifying a request to access an IP address. *See* Claim 1(b) (§VII.A.7 above). Furthermore, Fages teaches a first browser instance of the first browser type. *Id.*

8. Claim 7(a)(iii): "determining that the IP address is associated with the second content class such that the IP address should not be accessed by the first browser instance and confirming that a separate browser instance is not already in use and available to access the request, by:"

391. *See* Claim limitations 1(c)-1(d) (§§VII.A.8–VII.A.9 above), and 7(a)(ii) (§IX.A.7 above).

9. Claim 7(a)(iii)(1): "implementing a daemon, the daemon comprising a multi-threaded program executed outside the first browser instance that tracks active browser instances and launches a new browser instance in order to manage according to the rules,"

392. *See* Claim limitation 1(d)(i) (§VII.A.10 above) and Claim 6 (§IX.A.3 above, multi-threaded program).

10. Claim 7(a)(iii)(2): "receiving the request via a browser extension associated with the first browser instance,"

393. *See* Claim 1(d)(ii) (§VII.A.11 above).

11. Claim 7(a)(iii)(3): "forwarding the request from the browser extension to the daemon using domain sockets; and"

394. *See* Claim limitations 1(d)(ii) – 1(d)(iii) (§§VII.A.11–VII.A.12 above), and Claim 6 (§IX.A.3 above, domain sockets).

12. Claim 7(a)(iv): "generating a second browser instance to access the IP address, the second browser instance being of the second browser type."

395. *See* Claim limitations 1(e) (§VII.A.14 above) and 7(a)(ii) (§IX.A.7 above).

396. A POSITA would have understood that Fages's "new browser" (Claim 1(e), §VII.A.14 above) is a second browser instance of the second browser type. Indeed, Fages teaches launching a second browser to handle the sensitive URLs. Thus, a second browser instance would have been generated to access the IP address, where the second browser instance is of the second browser type (for handling sensitive content).

13. Claim 8: The method of claim 7, wherein the second browser instance is launched automatically to a client device for display.

397. *See* Claim 2 (§VII.A.15 above), and Claim 7 (§§IX.A.4–IX.A.12 above).

14. Claim 12(Pre): "A non-transitory computer readable medium encoded with a set of instructions for facilitating different categories of content to be displayed in different independent browser instances, the set of instructions, executable by a processor, comprising:"

398. *See* Claims 1(Pre) (§VII.A.5 above) and 7(Pre)-(a) (§§IX.A.4–IX.A.5 above). Fages alone or in view of Korycki teaches a processor and non-transitory computer readable medium encoded with a set of instructions.

15. Claim 12(a): "accessing data associating a first web content class with a first browser type and a second web content class with a second browser type;"

399. *See* Claim 7(a)(i) (§IX.A.6 above).

16. Claim 12(b): "accessing a request to access web content via a first browser instance, the first browser instance being of the first browser type; and"

400. *See* Claim 7(a)(ii) (§IX.A.7 above).

17. Claim 12(c): "comparing the web content to the first web content class and the second web content class to determine whether the web content is accessible to the first browser instance, by"

401. *See* Claims 1(c) (§VII.A.8 above) and 7(a)(iii) (§IX.A.8 above).

18. Claim 12(c)(i): "implementing a daemon, the daemon comprising a multi-threaded program executed outside the first browser instance that tracks active browser instances and launches a new browser instance in order to manage according to the rules,"

402. *See* Claim 7(a)(i) (§IX.A.6 above), 7(a)(iii)(1) (§IX.A.9 above).

19. Claim 12(c)(ii): "receiving the request via a browser extension associated with the first browser instance,"

403. *See* Claim 7(a)(iii)(2) (§IX.A.10 above).

20. Claim 12(c)(iii): "forwarding the request from the browser extension to the daemon using domain sockets."

404. *See* Claim 7(a)(iii)(3) (§IX.A.11 above).

21. Claim 13(Pre): "The non-transitory computer readable medium of claim 12, further comprising:"

405. *See* §IX.A.14 above, Ground 3, Claim 12(Pre).

22. Claim 13(a): "generating a second browser instance to access the web content,"

406. *See* Claims 1(e) (§VII.A.14 above) and 7(a)(iv) (§IX.A.12 above).

23. Claim 13(b): "wherein the second browser instance is launched automatically to a client device for display."

407. *See* Claims 2 (§VII.A.15 above) and 8 (§IX.A.13 above).

24. Claim 16: The non-transitory computer readable medium of claim 12, wherein the second web content class is associated with sensitive information indicating that accessing the web content by the first browser instance presents a security risk.

408. *See* Claims 1(a) (§VII.A.6 above) and 12 (§§IX.A.14–IX.A.20 above).

25. Claim 17: The non-transitory computer readable medium of claim 12, wherein the first web content class is associated with a first plurality of URLs and the second web content class is associated with a second plurality of URLs, the first plurality of URLs being different from the second plurality of URLs.

409. *See* Claims 1(a) (§VII.A.6 above) and 12 (§§IX.A.14–IX.A.20 above).

26. Claim 18: The non-transitory computer readable medium of claim 12, wherein the data further defines rules limiting web access by the first browser instance to URLs defined by the first web content class.

410. *See* Claims 7(a)(i) (§IX.A.6 above) and 12 (§§IX.A.14–IX.A.20 above).

X. OPINION: GROUND 4

A. Obviousness over Fages, Korycki, Jai-Hayden, O'Niell, Cardona, and Russello

411. Claims 9-10 and 14-15 would have been obvious over Fages, Korycki, Jai, Hayden, O'Niell, Cardona, and Russello.

1. Claim 9: The method of claim 7, further comprising: executing an API call configured to intercept a system call to a web server associated with the IP address, the system call initiated by the first browser instance.

412. *See* §VIII.A.2 above, Ground 2, Claim 3, and §IX.A.7 above, Ground 3, Claim 7(a)(ii).

2. Claim 10: The method of claim 7, further comprising: executing a tracer to determine whether or not a requested network object is allowed to be accessed by the first browser instance.

413. *See* §VIII.A.6 above, Ground 2, Claim 5.

3. Claim 14(Pre): "The non-transitory computer readable medium of claim 12, further comprising:"

414. *See* §IX.A.14 above, Ground 3, Claim 12(Pre).

4. Claim 14(a): "executing an API call configured to intercept a system call to a web server,"

415. *See* §VIII.A.2 above, Ground 2, Claim 3, and §§IX.A.14–IX.A.20 above, Ground 3, Claim 12.

5. Claim 14(b): "the system call associated with the first browser instance."

416. *See* §VIII.A.2 above, Ground 2, Claim 3, and §§IX.A.14–IX.A.20 above, Ground 3, Claim 12.

6. Claim 15: The non-transitory computer readable medium of claim 12, further comprising: executing a tracer to determine whether or not the first browser instance is authorized to access a requested network.

417. *See* §VIII.A.6 above, Ground 2, Claim 5, §X.A.2 above, Claim 10, and §§IX.A.14–IX.A.20 above, Claim 12.

XI. OPINION: GROUND 5

A. Obviousness over Fages, Korycki, Jai, Hayden, O'Niell, Cardona, and Tazuma

418. Claim 11 would have been obvious over Fages, Korycki, Jai, Hayden, O'Niell, Cardona, and Tazuma.

1. Overview of Tazuma

419. U.S. Patent No. 7,493,414 to Tazuma ("Tazuma") was filed on April 19, 2005 and issued on February 17, 2009 (EX1010, cover).

420. Tazuma teaches generating a domain name system (DNS) query to resolve an IP address. Tazuma (EX1010), 8:38-49 ("With further reference to FIG. 2A, operation 112 includes the client 14 performing a Domain Name Service (DNS)

query to determine the IP address of the destination server. DNS is a standard IP protocol and service that can be used to translate domain names (e.g., www.yahoo.com, etc.) to IP addresses (e.g. 200.35.53.2, etc.). Because DNS servers can be chained or linked together, resolving a single DNS query may require the cooperation of one or more different DNS servers. The destination server can be any of a wide range of computing devices capable of servicing standard access requests from client computers, such as HTTP requests, DNS queries, web requests, IP requests, etc.").

421. Tazuma also teaches redirecting a browser in a network by rewriting an IP address in a DNS response when the requested IP address is not allowed to be accessed. *Id*., 1:14-24, 1:51-58, 2:35-50, 3:13-24, 4:40-52, 6:66-7:16, 7:8-11 (in this example, Tazuma uses a "urlredirector" to redirect URL requests: "URLredirector also participates in performing access control by using ACL 54 to determine whether to allow a URL to be retrieved by a browser or to redirect the browser instead."), 9:40-62, 12:1-7 (teaching Access Control List (ACL)-based redirection), 22:20-27, 22:58-66.

2. Claim 11(Pre): "The method of claim 7, further comprising:"

422. *See* Claim 7(Pre) (§IX.A.4 above).

3. Claim 11(a): "generating a domain name system (DNS) query to resolve the IP address;"

423. *See* Claim 7(a)(ii) (§IX.A.7 above). A POSITA would have understood that Fages would have generated a DNS query to resolve the IP address of the URL in its URL list that the user has selected to access. Indeed, a POSITA would have understood that generating a DNS query to resolve an IP address was a standard IP protocol and service and would have necessarily been present in Fages. *See* §VI.D above.

a. Tazuma's DNS Query

424. To the extent it is argued that Fages does not teach generating a DNS query to resolve the IP address, Tazuma teaches such. Tazuma teaches a computer "performing a Domain Name Service (DNS) query to determine the IP address of the destination server. DNS is a standard IP protocol and service that can be used to translate domain names (e.g., www.yahoo.com, etc.) to IP addresses (e.g., 200.35.53.2, etc.)." Tazuma (EX1010), 8:38-49. A POSITA would have understood that this would have been a standard method for generating a DNS query to resolve an IP address. *See* §VI.D above.

425. **In sum**, the Fages-Korycki-Jai-Hayden-O'Niell-Cardona-Tazuma combination teaches "generating a domain name system (DNS) query [Tazuma's DNS queries] to resolve the IP address;"

b. Motivation to Combine Fages-Korycki-Jai-Hayden-O'Niell-Cardona with Tazuma (DNS Query)

426. A POSITA would have been motivated to modify Fages-Korycki-Jai-Hayden-O'Niell-Cardona with Tazuma to advantageously generate a DNS query to resolve an IP address. While a POSITA would have understood that Fages would have operated in this standard computer networking way, Fages itself does not expressly teach generating a DNS query to resolve an IP address. Tazuma, however, does teach this well-known technique in computer networking. This use of a known technique would have improved the modified Fages's similar computer system in the same way as Tazuma's computer system. Indeed, using a DNS query to resolve an IP address means a user does not have to enter a numerical IP address to access a website (*e.g.*, a user enters www.yahoo.com in the web browser, which is easier for people to remember, than entering instead 200.35.53.2, such as in Tazuma's example) because the DNS query translates the URL into the appropriate IP address.

427. Furthermore, this use of a known technique would have resulted in functional DNS queries to translate URLs including domain names into IP addresses, resulting in a functional computer system to access websites on the Internet, for example, in a well-known manner. Indeed, Tazuma expressly teaches that a DNS query is part of the IP protocol, which a POSITA would have been familiar with, and Tazuma teaches using an Access Control List (ACL) to determine if a user may access particular websites. Tazuma (EX1010), 8:38-43, 11:59-64 (teaching parsing

URLs into each part such as the protocol name, server name, and path string), 12:1-7, 14:40-55, 22:20-27, 22:58-66, 6:35-51, 7:6-11.

428. A POSITA would have had a reasonable expectation of success when modifying the Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination with Tazuma because Tazuma teaches a standard DNS query technique used in the IP protocol for networking (*e.g.*, browsing the Internet) and because the Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination's computer system is similar to Tazuma's, including having Internet access and using lists to control URLs that may be visited (*e.g.*, Access Control List (ACL)). *Id*., 7:6-11, 12:1-7. For example, Fages teaches a server computer that executes web browsers and accesses web content via network connections (*e.g.*, Internet) with web servers. *See* Claim 1 (§§VII.A.5–VII.A.14 above).

429. Tazuma similarly teaches a standard computer interacting with web servers. Tazuma (EX1010), 5:10-37, 7:17-37, 8:13-60, 6:9-51. A POSITA would have understood that modifying the Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination would have required only straightforward modifications to programming instructions, which would have been well-within the skillset of a POSITA. For example, Tazuma teaches implementing its invention in programming instructions. *Id*., 7:17-23, 8:13-26.

4. Claim 11(b): "determining that the IP address is associated with the second content class; and"

430. *See* Claim 7(a)(iii) (§IX.A.8 above).

5. Claim 11(c): "issuing a DNS response, the IP address being rewritten in the DNS response to prevent the first browser instance from accessing the IP address and information of the second content class; and"

431. *See* Claim 7(a)(iii) (§IX.A.8 above). Fages teaches utilizing multiple independent browsers where different browsers and sessions are related to first and second content classes. *See* Claims 1 (§§VII.A.5–VII.A.14 above) and 7 (§§IX.A.4–IX.A.11 above).

a. '721 Patent DNS Response

432. The '721 Patent describes rewriting an IP address in a DNS response when the requested IP address is not allowed to be accessed. '721 Patent (EX1001), 7:36-44.

b. Tazuma's DNS Response

433. Tazuma teaches redirecting a browser in a network by "changing an original destination address to another address in such a way that the client is not aware of the change." Tazuma (EX1010), 4:40-52. Thus, Tazuma teaches redirecting/modifying (*rewriting*) an IP address in a DNS response when the requested IP address is not allowed to be accessed. *Id.*, 1:14-24, 1:51-58, 2:35-50, 3:13-24, 4:40-52, 6:66-7:16, 7:8-11 (in this example, Tazuma teaches using a "urlredirector" to redirect URL requests: "URLredirector also participates in performing access control by using ACL 54 to determine whether to allow a URL to

be retrieved by a browser or to redirect the browser instead."), 9:40-62, 12:1-7 (teaching Access Control List (ACL)-based redirection), 22:20-27, 22:58-66.

434. Indeed, I already opined in §XI.A.3 above for Claim 11(a) regarding ACL redirection and how this DNS redirection is a similar technique to what Fages and Korycki use. *See* Claim 11(a) (§XI.A.3 above). Thus, a POSITA would have understood that one way Tazuma teaches redirection is by rewriting the URL or IP address to redirect the browser to a different webpage than the original one a user requested. A POSITA would have understood that redirection in this manner prevents a browser from accessing the requested IP address or URL because the browser must access the provided (redirected) URL. Tazuma (EX1010), 1:20-24, 1:51-58, 4:40-52.

435. **In sum**, Fages-Korycki-Jai-Hayden-O'Niell-Cardona-Tazuma teaches "issuing a DNS response [Tazuma's DNS response], the IP address being rewritten in the DNS response [Tazuma's rewriting technique] to prevent the first browser instance from accessing the IP address and information of the second content class."

c. Motivation to Combine Fages-Korycki-Jai-Hayden-O'Niell-Cardona with Tazuma (DNS Redirection)

436. A POSITA would have been motivated to combine Fages-Korycki-Jai-Hayden-O'Niell-Cardona with Tazuma to advantageously perform redirection and achieve Tazuma's benefits. For example, redirection would have amounted to using a known technique of returning a rewritten IP address or URL in response to a

request being unauthorized or unable to be fulfilled instead of simply returning no response at all (or returning an error page). Tazuma (EX1010), 1:59-61, 4:53-5:8, 8:61-9:16, 9:40-62, 16:18-24, 18:40-62, 19:36-50. This use of a known technique would have improved the similar Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination's computer system and Tazuma's computer system in the same way, including because both Fages and Tazuma teach web browsers and URL lists (*e.g.*, ACL).

437. For example, a POSITA would have understood and been motivated to utilize redirection to provide an explanation to a user as to why the browser instance such as S_4 is prevented from accessing an IP address in secure/sensitive URL lists L_{i1} instead of simply blocking the attempt (or returning an error web page). This additional information would help inform and educate a user for future use to know which web browser instance to utilize to (safely) go to an online banking website, for example. In one example, a POSITA would have found it advantageous to inform a user that the browser in session S_4 through which the user has requested the URL/IP address is authorized to view only "non-sensitive"/"non-secure" content not in any of the sensitive/secure URL lists (ACL list) L_{i1}. *See* Claims 1 (§§VII.A.5–VII.A.14 above) and 7 (§§IX.A.4–IX.A.12 above). Thus, a user would have been informed that the web browser will not browse "sensitive" URLs (*e.g.*, banking).

Using Tazuma's redirection technique would thus have advantageously informed a user instead of simply denying the request.

438. A POSITA would have had a reasonable expectation of success when modifying the Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination with Tazuma. Indeed, this modification would have required only straightforward changes to programming instructions, which would have been well-within the skillset of a POSITA. As I discussed in §XI.A.3 above, Tazuma teaches implementing several aspects of its invention in computer programming instructions, which would have led to a reasonable expectation of success when modifying Fages. *See* Claim 11(a) (§XI.A.3 above).

439. Also discussed in §§VII.A.5–VII.A.14 above, Fages teaches a native method for loading programming instructions to the server computer, further leading to a reasonable expectation of success when modifying programming instructions. *See* Claim 1 (§§VII.A.5–VII.A.14 above).

440. Furthermore, similar to Tazuma, Fages teaches performing an ACL type security by comparing requested URLs with URLs on access lists. *See* Claim 1 (§§VII.A.5–VII.A.14 above).

441. Additionally, Korycki expressly teaches using ACL list security to verify if a user may access a web resource (*see* Claim 1, §§VII.A.5–VII.A.14 above). Thus, in the Fages-Korycki-Jai-Hayden-O'Niell-Cardona-Tazuma combination,

adding redirection in addition to ACL (or similar URL list-based security, such as Fages's native security) would not have required adding in additional structures or infrastructure. Thus, modifying the Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination per Tazuma would have required only adding redirection functionality as per Tazuma.

442. Lastly, a POSITA would have had a reasonable expectation of success because Tazuma teaches redirection functionality already existing before Tazuma, indicating to a POSITA that redirection functionality was well-known in the prior art. Tazuma (EX1010), 2:35-50 ("using current redirection methods."). Indeed, Tazuma teaches improving on the then-current redirection methods, indicating that URL redirection was already a known concept in the prior art.

6. Claim 11(d): "returning the IP address as rewritten to the first browser instance."

a. Tazuma's Returning a Rewritten IP Address

443. The Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination as modified with Tazuma teaches returning the IP address as rewritten to the first browser instance.

444. I note that this is fundamentally what URL redirection accomplishes: a requested URL may be redirected to a different URL, and the rewritten/different URL is what is returned to the user. For example, Tazuma teaches "in operation 456 the URL requested is checked against the ACL. If the URL passes the ACL check,

in operation 460 WRH 42 performs the request on behalf of the browser, returning any reply back to the browser. **If the URL is determined to be restricted, in operation 464 a reply is sent to the browser causing it to be redirected to a pre-designated redirection URL**, e.g., a splash or login page."). Tazuma (EX1010), 22:21-27, 22;58-66, 4:40-52 ("A common form of IP redirection is to change an original destination address to another address in such a way that the client is not aware of the change."), 1:21-24 ("Web browser redirection can be used to intercept an access request to load an original destination address (e.g., uniform resource locator (URL) or web address, etc.) and then redirect the web browser, for example, to a specific portal or local web page.").

445. Thus, in my opinion, the Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination as modified by Tazuma teaches returning an IP address as rewritten to the first browser, for example, when a user requests a sensitive URL with a browser instance that is designated for non-sensitive content only. Instead of returning the requested sensitive URL (*e.g.*, banking), a rewritten URL is returned (*e.g.*, a message indicating why the request was not completed), and a second browser instance may be launched to handle the requested sensitive URL. *See* Claim 1 (§VII.A.5–VII.A.14).

446. **In sum**, Fages-Korycki-Jai-Hayden-O'Niell-Cardona-Tazuma teaches "returning the IP address as rewritten [Tazuma's rewriting technique] to the first browser instance."

b. Motivation to Combine Fages-Korycki-Jai-Hayden-O'Niell-Cardona with Tazuma (DNS Redirection)

447. A POSITA would have been motivated to modify the Fages-Korycki-Jai-Hayden-O'Niell-Cardona combination and would have had a reasonable expectation of success in doing so for the same reasons discussed in §XI.A.5 above and applied herein. *See* Claim 11(c) (§XI.A.5 above).

XII. CONCLUSION

448. I declare that all statements made herein of my knowledge are true, and that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: September 12, 2025 By:

Erez Zadok, PhD