



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
90/011,521	03/01/2011	6,192,476	13557.105128	8619

25226 7590 12/20/2011

MORRISON & FOERSTER LLP
755 PAGE MILL RD
PALO ALTO, CA 94304-1018

EXAMINER

ART UNIT PAPER NUMBER

DATE MAILED: 12/20/2011

Please find below and/or attached an Office communication concerning this application or proceeding.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

DO NOT USE IN PALM PRINTER

(THIRD PARTY REQUESTER'S CORRESPONDENCE ADDRESS)

Robert T. Neufeld, Esq.
KING & SPALDING
1180 Peachtree Street, N.E.
Atlanta, Georgia 30309-3521

EX PARTE REEXAMINATION COMMUNICATION TRANSMITTAL FORM

REEXAMINATION CONTROL NO. 90/011,521.

PATENT NO. 6,192,476.

ART UNIT 2173.

Enclosed is a copy of the latest communication from the United States Patent and Trademark Office in the above identified *ex parte* reexamination proceeding (37 CFR 1.550(f)).

Where this copy is supplied after the reply by requester, 37 CFR 1.535, or the time for filing a reply has passed, no submission on behalf of the *ex parte* reexamination requester will be acknowledged or considered (37 CFR 1.550(g)).

Office Action in Ex Parte ReexaminationControl No.
90/011,521Patent Under Reexamination
6,192,476Examiner
DENNIS BONSHOCKArt Unit
2173

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

- a ☒ Responsive to the communication(s) filed on 16 September 2011. b ☒ This action is made FINAL.
c ☐ A statement under 37 CFR 1.530 has not been received from the patent owner.

A shortened statutory period for response to this action is set to expire 2 month(s) from the mailing date of this letter. Failure to respond within the period for response will result in termination of the proceeding and issuance of an *ex parte* reexamination certificate in accordance with this action. 37 CFR 1.550(d). **EXTENSIONS OF TIME ARE GOVERNED BY 37 CFR 1.550(c).** If the period for response specified above is less than thirty (30) days, a response within the statutory minimum of thirty (30) days will be considered timely.

Part I THE FOLLOWING ATTACHMENT(S) ARE PART OF THIS ACTION:

1. ☐ Notice of References Cited by Examiner, PTO-892. 3. ☐ Interview Summary, PTO-474.
2. ☒ Information Disclosure Statement, PTO/SB/08. 4. ☐ _____

Part II SUMMARY OF ACTION

- 1a. ☒ Claims 1-21 are subject to reexamination.
1b. ☐ Claims _____ are not subject to reexamination.
2. ☐ Claims _____ have been canceled in the present reexamination proceeding.
3. ☒ Claims 8,9,17 and 18 are patentable and/or confirmed.
4. ☒ Claims 1-7,10-16 and 19-21 are rejected.
5. ☐ Claims _____ are objected to.
6. ☐ The drawings, filed on _____ are acceptable.
7. ☐ The proposed drawing correction, filed on _____ has been (7a) ☐ approved (7b) ☐ disapproved.
8. ☐ Acknowledgment is made of the priority claim under 35 U.S.C. § 119(a)-(d) or (f).
 a) ☐ All b) ☐ Some* c) ☐ None of the certified copies have
 1 ☐ been received.
 2 ☐ not been received.
 3 ☐ been filed in Application No. _____.
 4 ☐ been filed in reexamination Control No. _____.
 5 ☐ been received by the International Bureau in PCT application No. _____.
 * See the attached detailed Office action for a list of the certified copies not received.
9. ☐ Since the proceeding appears to be in condition for issuance of an *ex parte* reexamination certificate except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte* Quayle, 1935 C.D. 11, 453 O.G. 213.
10. ☐ Other: _____

cc: Requester (if third party requester)

Final Action

Reexamination

This is an ex parte reexamination of U.S. Patent Number: 6,192,476. Patent claims 1-7, 10-16, and 19-21 are under reexamination. Claims 8, 9, 17, and 18 are confirmed. This is a final action in response to the request for reconsideration filed 9-16-2011.

Affidavits

Affidavits A-O have been considered and placed on the record. Further comments regarding content of the Affidavits can be found below in the Arguments section.

Prior Art

Claims 1-7, 10-16, and 19-21 are reexamined on the basis of the following references:

Fischer (*U.S. Patent Number: 5,412,717*)

Griffin (*U.S. Patent Number: 5,958,050*)

Chan (*The Java Class...*)

Claim Rejections - 35 USC § 102

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

Art Unit: 2173

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

Claims 1-7, 10-16, and 19-21 are rejected under 35 U.S.C. 102(b) as being anticipated by USPN 5,412,717 to Fischer. See Request 01/10/2011, pages 16-18 and Exhibit 10 Claim Chart.

Claims 8-9 and 17-18 are not included in the anticipation rejection. Fischer does not disclose **the step of setting a flag associated with said first routine to indicate that said first routine is privileged**. Fischer does not teach a flag field.

Gong '476 describes a flag at FIG. 3 and 3: 33-40, "A privileged routine is allowed to perform certain actions even if the routine that called the privileged routine does not have permission to perform those same actions...a flag in a frame in the calling hierarchy corresponding to a privileged routine is set to indicate that the privileged routine is privileged..." Also see Gong '476, 14: 56,- 15: 14, "One technique to track which invocation of a particular method are enabling invocations is to set a flag (privilege flag 312) in the frame 310 corresponding to each enabling invocation...each frame has a privilege flag value..."

See Gong '476 3: 33-41, "...certain routines may be 'privileged'. A privileged routine is allowed to perform certain actions even if the routine that called the privileged routine does not have permission to perform those same actions...a flag in a frame in the calling hierarchy corresponding to a privileged routine is set to indicate that the privileged routine is privileged."

While Fischer '717 does recite (12: 58-65), "The present invention, while it primarily focuses on defining functions which restrict the ability of a program to access...could also...be used to extend the capabilities beyond those normally allowed...could be allowed to perform extended functions (privileged routines)." *Fischer* (10:47-49) discloses the field 156 of the program control block identifies the location in storage (157) of one or more the PAIs... Examiner asserts that such a location pointer does not function as would the "flag" of the instant limitation.

Per **claim 1**, Fischer discloses a method for **providing security**. The Fischer disclosure references 'authorization entries' within the "program authorization information," or "PAI," to verify access authority to other code and resources. "... providing enhanced computer system security while processing computer programs... " Fischer at 1:20-25.

Fischer discloses **detecting when a request for an action is made by a principal**.

Gong '476 describes a 'principal' at 2: 37-39, "A 'principal' is an entity in the computer system to which permissions are granted. Examples of principals include processes, objects and threads. A 'permission' is an authorization by the computer system that allows a principal to perform a particular action or function." 3: 67 – 3:2, "...access rights for a principal are determined dynamically based on the source of the code that is currently being executed by the principal (e.g., thread, process). Therefore the "program" defined in Fischer is clearly a process, executable set of code, ran by the

Art Unit: 2173

computer system (see column 9, line 19 through column 10, line 7) and requesting usage of the computers resources (see column 10, lines 7-52).

Broadly a 'principal' reads on Fischer's 'system monitor' (2: 17-18) which limits the ability of a program to be executed. See 10: 10-13, "The program control block 140 is the data structure utilized by the system monitor to control the execution of an associated program. Fischer (15: 56-58) also uses the term 'supervisor program' for controlling the processing of a program being executed.

See FIG. 10 (#300, call program X). Fischer inquiries into the permissions of a given principal's PAI upon the principal's request for an action: "When the program is to perform a function or access a resource (detecting when request for action is made), the associated PAI (PAI associated with the calling program) is monitored to confirm that the operation is within the defined program limits. If the program (calling program) attempts to do anything outside the authorized limits, then the program execution is halted." *Id.* at 2:43-48.

Fischer (2: 24-31; 7: 14-15), "The set of authorities and/or restrictions assigned to a program to be executed are referred to herein as "program authorization information" (or "PAI")...thereafter associated with each program to be executed."

Fischer discloses a “program” that executes “operations” (2: 35) or performs a “function (2: 43-44) (by way of a thread of execution / a ‘principal’).” Fischer discloses the program as part of data objects (object oriented language, where instances of a class are instantiated as an object; See 4: 11, object oriented). Fischer discloses (10: 3-57) a program (“originating program”) that calls a second “program.” The second program, in turn calls a third program, such that called programs are hierarchically executed. Each program, the ‘originating program’ or authorized subsequently called program, places a ‘program control block’ (PCB) (10: 36-37) on the execution stack (stack/call stack; load program X), following a ‘yes’ exit from #322 or #320.

Fischer discloses in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein said permissions are associated with said plurality of routines based on a first association between protection domains and permissions.

(9: 58-63), “After the PAI has been assigned, any time the system runs the associated program (by the principal), the system software...ensures that the program safely runs in a manner consistent with the PAI. Thus, the program has been effectively placed in a ‘safety box’ (124).” (determining whether said action is authorized based on permissions...) (9: 17 – 10: 23), “The program control block 140 is loaded with program

Art Unit: 2173

authorization information such that the PAI can be readily referenced as the associated program is executed so as to ensure that the program performs functions and accesses resources in conformance with its assigned authorizations. The program control block associated with the program to be executed is located in a storage area which cannot be modified by the program." (15: 56- 16: 2), "FIGS. 10 and 11 illustrate the sequence of operations of a supervisor program for controlling the processing of a program being executed in accordance with its program authorization information. The processing of a program 'X' and its program authorization information illustrated in FIG. 10 is initiated while the computer is executing a supervisor routine (supervisor program executing supervisor routine, i.e., principal, controls the processing of a called program 'X', i.e., nested programs / calling hierarchy). As shown in FIG. 10 at 300, a calling program calls program 'X' for execution (i.e., supervisor program calls program 'X'). Thereafter, a program control block is created for program X. The program control block created will not be added to the top of the execution stack until it is determined that the program is permitted to be invoked and verification is successful completed (permissions associated with routines of calling program based on PAI defined for the calling program and permissions/ authorization entries defined within the PAI. See 'authorization entries' of PAI at 5: 55- 61) Thus, if the program fails a security check, it will not be placed in the program execution chain." (emphasis added)

Here the program further requests calls to other program and other functions of varying permission levels (see column 10, lines 7-52 and column 19, lines 5-15); and has associated with it a plurality of routines in a hierarchy, each level having an

Art Unit: 2173

associated program control block (PCB) that links to parent or child levels with their own PCBs and lists program authorization information (PAI) for the particular sub-program being executed (see column 10, lines 23-57 and figure 5).

The term 'protection domains' is defined by Gong '476 at 8: 55-64, "...protection domains are used to enforce security within computer systems. A protection domain can be viewed as a set of permissions granted to one or more principals. A permission is an authorization by the computer system that allows a principal to execute a particular action or function...permissions involve an authorization to perform an access to a computer resource..." Broadly Fischer's PAI ('717, 2: 34-36) is reads on the claimed 'protection domain.'

Fischer also incorporates by reference (6: 37) two patents which disclose data hierarchies: USPN 4,868,877 and USPN 5,005,200. The two patents disclose an enhanced digital signature certification which employs a "hierarchy of nested certifications and signatures" which fairly reads on a calling hierarchy (nested certificate hierarchy must be examined for permissions) associated with said principal.

Fischer discloses ('717, 6: 25-58) the **association between permissions and the plurality of routines based on a first association between protection domains and permissions**. See '877, 13: 50-51, hierarchy of authority; '877, 15: 18-20, hierarchy of all certificates; '877, 17: 26-27, authorized by antecedent certificates. See '200, 7: 3-7,

Art Unit: 2173

certificates created convey authorizations, restrictions; '200, 7: 17-18, hierarchically derived certificates; '200, 10: 50-67, digital signature accompanied by at least one valid certificate, may be associated with one or more other valid certificates hereinafter referred to as antecedents to that certificate; '200, 20: 10-12, B's signature and the hierarchy of all certificates and signatures which validate it are kept by A and sent along whenever A uses this certificate; '200, 22: 17-23, all certificates must be accompanied by signatures which themselves are authorized by antecedent certificates. The digital certificates and signatures take into account permissions of programs other than the requesting program when determining whether a requested act may be performed. Thus Fischer's teachings are not limited to the one and only program (the requesting program) located at the top of the call stack (execution stack), but actually require consideration of permissions associated with a plurality of routines in the calling hierarchy of the digital signature / certificate authorizations.

It is noted that for the purpose of reexamination the term 'routine' includes within its scope such terms as function, operation, program, method, or the concept of "access" to a resource.

Per **claim 2**, Fischer discloses **the step of detecting when a request for an action is made includes detecting when a request for an action is made by a thread**. Fischer's invention inquires into the permissions of a given principal's PAI upon the principal's request for an action: "When the program is to perform a function or access a resource, the associated PAI is

Art Unit: 2173

monitored to confirm that the operation is within the defined program limits. If the program attempts to do anything outside the authorized limits, then the program execution (thread of execution) is halted." *Id.* at 2:43-48. See FIGs 3A – 3D, describing authorizations found in PAIs associated with programs to be executed by thread.

See Gong '476, 2: 36- 3: 2, "A 'principal' is an entity in the computer system to which permissions are granted...principals include processes, objects, and threads." The 'principal' performs actions or functions (i.e., code is executed by the 'principal').

See FIG. 10, #300 Call Program X and '717, 15: 56-63. At this point the request for an action by the executing thread is detected. A supervisor program for controlling the processing of a program being executed, executing its supervisor routine (thread of execution), detects a call to program X (detects an action).

Fischer discloses the step of determining whether said action is authorized includes determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said thread.

Fischer discloses determining whether said action is authorized includes determining whether said action is authorized based on an association between permissions and a

Art Unit: 2173

plurality of routines in a calling hierarchy in view of associating permissions by aggregating constraints of PAIs associated with program control blocks on the execution / call stack or by associating with the nested certifications and signature routines in a calling hierarchy associated with a thread, as taught by USPN 4,868,877 and USPN 5,005,200.

See Fischer FIG. 5 and 10: 24-26, "FIG. 5 is an illustration of a program control block (PCB) data structure 140 in accordance with an exemplary embodiment of the present invention. The program control block 140 is the data structure utilized by the system monitor to control the execution of an associated program.

The program control block 140 is loaded with program authorization information such that the PAI can be readily referenced as the associated program is executed so as to ensure that the program performs functions and accesses resources in conformance with its assigned authorizations. The program control block associated with the program to be executed is located in a storage area which cannot be modified by the program.

As shown in FIG. 5, an originating program (whose PCB is identified at 180) calls a program (having a PCB 170) which will, in turn, will call the program 140 is shown in detail in FIG. 5. Each new PCB will include a field such as 150 that points to the 'previous' or calling program control block. A field may also be utilized to identify the 'next' program control block file.

"Thereafter, the program X's program authorizing information is combined, as appropriate, with the PAI associated with the PCB of the calling program (association between permissions and a plurality of routines in a calling hierarchy associated with said thread), if any. This combined PAI, which may include multiple PAIs, is then stored in an area of storage which cannot generally be modified by the program and the address of the PAI is stored in the process control block (PCB) as indicated in field 156 of FIG. 5. Thus, if program X is called by a calling program, it is subject to all its own constraints as well as being combined in some way with the constraints of the calling program, which aggregate constraints are embodied into program X's PAI (determine whether action is authorized). In this fashion, a calling program may not be permitted to exceed its assigned bounds by merely calling another program. There are many alternative ways that a program's PAI could be combined with the PAI of the program which invokes it—depending on the strategies which are applicable to the current environment, and the inherent nature of the programs themselves. It may even be likely that even the method of combination is itself one of the PAI authorities, or qualifiers, of either or both the invoking or invoked program.

For example, it is reasonable to restrict a called program to the lesser of its 'normal' PAI authority and that of its calling program--to ensure the calling program cannot mischievously misuse the called program's greater authority to circumvent its own limitations.

Art Unit: 2173

On the other hand, for called programs which carefully verify their own actions, it could be possible to allow the called program greater inherent authority than the program which calls it this way sensitive resources could be made available to wider use by mediating such use through trusted sub-programs. The possibilities for such combination must be carefully considered, not only by the designers of the underlying control system, but also by those who assign authority to each program. Thereafter, the program is loaded and the hash of the program is computed based on the algorithm specified in the program's PAI." Fischer at 17:40-18:10. (emphasis added)

Alternately, consider Fischer's incorporation by reference (Fischer '717, 6:37) of two patents which explicitly disclose data calling hierarchies: USPN 4,868,877 and USPN 5,005,200. See USPN 5,005,200, Abstract, for disclosure of an "enhanced digital signature certification" employing "[a] hierarchy of nested certifications and signatures." The '877 and '200 patents disclose a plurality of routines in a calling hierarchy associated with a thread. An examination of the nested certificates determines whether the action is authorized.

Per **claim 3**, Fischer discloses **the calling hierarchy includes a first routine**. For example, Fischer discloses (10: 24-27) calling a plurality of routines: "As shown in FIG. 5, an originating program (whose PCB is identified at 180) (first routine) calls a program (having a PCB 170) which will, in turn, will call the program 140 is shown in detail in

FIG. 5." The called programs associated with PCBs 180, 170, and 140 are representative of the calling hierarchy.

Fischer discloses **the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with said first routine.**

Fischer utilizes the PAI data structure associated with a program to determine whether the requested action is authorized. The data structure includes a header segment 114 which, by way of example only, may define the type of object that follows, e.g., a purchase order related object or any other type of electronic digital object. The program authorization information is embedded in a segment 116 which specifies the authorization for the object's program or programs in a manner to be described more fully hereinafter." Fischer at 7:49-8:2. "FIGS. 10 and 11 illustrate the sequence of operations of a supervisor program for controlling the processing of a program being executed in accordance with its program authorization information. The processing of a program 'X' and its program authorization information illustrated in FIG. 10 is initiated while the computer is executing a supervisor routine. As shown in FIG. 10 at 300, a calling program calls program X for execution.

Thereafter, a program control block is created for program X. The program control block created will not be added to the top of the execution stack until it is determined that the program is permitted to be invoked and verification is successful completed (determining whether said action is authorized). Thus, if the program fails a security

Art Unit: 2173

check, it will not be placed in the program execution chain (will not be invoked / placed on the call stack). In addition to creating a 'tentative' program control block, the called program will be located through an appropriate program directory during the processing in block 302.

See FIG. 10 and steps between calling the program at #300 and invoking / loading the 'authorized' program onto the call stack (i.e., steps #302-#322 determine whether a permission required to perform said action is encompassed by at least one permission associated with said first routine).

Per claim 4, Fischer discloses the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy.

Fischer discloses (17: 40-51) an embodiment wherein PAI information is aggregated, and access is determined based on this aggregated grouping. See FIG. 5. An executing program is aware of the hierarchy of calling programs (i.e., program 180 calls program 170, which calls program 140, etc). Pointer fields (FIG. 5, #150) in each PCB of the execution stack are aware of the 'previous' or calling program control block and may also be utilized to identify the 'next' program control block file. Fischer at 10:23-39. "There are many alternative ways that a program's PAI could be combined with the PAI of the program which invokes it—depending on the strategies which are

Art Unit: 2173

applicable to the current environment, and the inherent nature of the programs themselves. It may even be likely that even the method of combination is itself one of the PAI authorities, or qualifiers, of either or both the invoking or invoked program. For example, it is reasonable to restrict a called program to the lesser of its 'normal' PAI authority and that of its calling program--to ensure the calling program cannot mischievously misuse the called program's greater authority to circumvent its own limitations. On the other hand, for called programs which carefully verify their own actions, it could be possible to allow the called program greater inherent authority than the program which calls it--this way sensitive resources could be made available to wider use by mediating such use through trusted sub-programs. The possibilities for such combination must be carefully considered, not only by the designers of the underlying control system, but also by those who assign authority to each program. Thereafter, the program is loaded and the hash of the program is computed based on the algorithm specified in the program's PAI." Fischer at 17:40-18:10.

Per **claim 5**, Fischer discloses **a method for providing security**. The Fischer disclosure employs "program authorization information," or "PAI," to limit a principal's access authority to other code and resources. "More particularly, the invention relates to a method and apparatus for providing enhanced computer system security while processing computer programs, particularly those of unknown origin, which are transmitted among users." Fischer at 1:20-25.

Art Unit: 2173

Fischer discloses **detecting when a request for an action is made by a principal.**

Fischer inquiries into the permissions of a given principal's PAI upon detecting the principal's request for an action: "When the program is to perform a function or access a resource (when request for an action is detected), the associated PAI is monitored to confirm that the operation is within the defined program limits. If the program attempts to do anything outside the authorized limits, then the program execution is halted." *Id.* at 2: 43-48. Therefore the "program" defined in Fischer is clearly a process, executable set of code, ran by the computer system (see column 9, line 19 through column 10, line 7) and requesting usage of the computers resources (see column 10, lines 7-52).

"FIGS. 10 and 11 illustrate the sequence of operations of a supervisor program (principal thread of execution) for controlling the processing of a program being executed in accordance with its program authorization information. The processing of a program 'X' and its program authorization information illustrated in FIG. 10 is initiated while the computer is executing a supervisor routine (principal). Fischer at 15: 56-16: 11.

Fischer discloses **determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said principal.** Fischer discloses "an originating program" that "calls a program" having a program control block, or PCB. Fischer at 10: 24-26; 10: 8-39;

FIG. 5. "Each new PCB will include a field... that points to the 'previous' of calling program control block." See *id.* at 10: 27-29. Fischer also notes (FIG. 10; 15: 56-59) the sequence of operations done by a "supervisor program (principal) for controlling the processing of a program being executed in accordance with its program authorization information." The steps of FIG. 10 prior to step 326 relate to "determining whether said action is authorized." If authorization is determined, the called program is loaded (invoked) onto the call stack at #326. A plurality of routines in a calling hierarchy are added to the top of the call stack (routines in a calling hierarchy associated with said principal; 16: 1-2, "placed in the program execution chain") and each routine is associated with permissions defined in its PAI. As an added routine completes its execution, it is removed from the call stack, and subsequently called routines will be placed on the call stack. Then, "[w]hen a called program finishes executing, the system removes its associated PCB from the top of the executed stack (call stack), removes the associated program from storage, removes the associated authorizing information and accesses the program control block immediately below it in the stack (call stack)." See *id.* at 10:31-36. See PAIs defined at 2: 16-48. PAI authorizing information of the program at the top of the call stack may be combined with the PAI associated with the program control block (PCB) of the calling program, or of multiple PAIs on the call stack. Aggregated constraints found in PAI at top of call stack provide an association between permissions and a plurality of routines. See *id.* at 17:40-56.

Here the program further requests calls to other program and other functions of varying permission levels (see column 10, lines 7-52 and column 19, lines 5-15); and

Art Unit: 2173

has associated with it a plurality of routines in a hierarchy, each level having an associated program control block (PCB) that links to parent or child levels with their own PCBs and lists program authorization information (PAI) for the particular sub-program being executed (see column 10, lines 23-57 and figure 5).

Fischer also incorporates by reference two patents which explicitly disclose data hierarchies. See *id.* at 6:37; 16: 12-65 (incorporating by reference U.S. Patent Nos. 4,868,877 and 5,005,200 disclosing an "enhanced digital signature certification" which employs "[a] hierarchy of nested certificates and signatures (trust level determined). See USPN 5,005,200 at Abstract. FIG. 10, steps 306, 308, 314, 328, 320, 324 relate to a plurality of nested routines required to confirm digital signatures. Checking the nested certificates for authorization is reads on claim language 'determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said principal.'

Fischer discloses **wherein each routine of said plurality of routines is associated with a class**. Fischer notes an object oriented approach throughout the '717 Specification. See FIG. 3C and related text at 7: 29-8: 44, "type of object," "object programs," "data associated with this instance of the object (an instance of a class is a class object)." As an example, Fischer discloses (8: 7-11), "The object program might

Art Unit: 2173

store...and send...(plurality of routines associated with object, where an object is an instance of a class)..."

Fischer discloses **wherein said association between permissions and said plurality of routines is based on a second association between classes and protection domains.**

Fischer discloses an object oriented format. An instance of an object (i.e., a program object) is derived from class template. "In one contemplated embodiment of the present invention, programs may be part of data objects, which are written in a high-level control language and are executed by a standardized interpreter program which executes this high-level language. In this case, part of the interpreter's task is to verify that the functions encountered in the high level logic are, in fact, permissible (i.e. verify associated permissions). If such tasks are not permissible, the interpreter then suppresses the execution of the program not authorized to perform such tasks." Fischer at 3:11-20.

A protection domain is represented by the PAI associated with a calling program. The PAI associates permission / authorities granted to the calling program. A calling program may have a plurality of routines, as discussed above. The class, from which the program object is derived, has an associated PAI. The program authorization information is embedded in a segment 116 which specifies the authorization for the object's program or programs in a manner to be

Art Unit: 2173

described more fully hereinafter." "In accordance with the present invention, a PAI is associated with programs (program objects instantiated from class objects) to be executed. FIGS. 3A through 3D depict four exemplary approaches for associating program authorization information with a program" Fischer at 7:14-18, 7:49-8:2.

As previously discussed, with properly granted permissions of a calling program, subsequent program objects are placed onto the top of the call stack / execution stack and the most recently added program object may be subject to the combined constraints (authorizations) of previously added program objects. "...a program's PAI could be combined with the PAI of the program which invokes it..." (based on a second association between classes and protection domains, where the 'second association' refers to programs objects derived from classes and their associated PAIs defining authorizations, that were previously placed on the call stack) Fischer at 17: 47-62.

Per **claim 6**, Fischer discloses **a method for providing security** by way of "program authorization information," or "PAI," to limit a principal's access authority to other code and resources. Fischer at 1:20-25; 2: 49-55.

Fischer discloses **detecting when a request for an action is made by a principal**.

See limitation addressed in rejection of claim 1 above.

Fischer discloses in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein a first routine in said calling hierarchy is privileged.

Access to a calling hierarchy is disclosed, as explained above with respect to claim 1.

The limitation of "**wherein a first routine in said calling hierarchy is privileged**" is a variation of the claim 1 limitation.

The Gong '476 specification describes a "privileged routine" ('476, 3: 33-37) as "allowed to perform certain actions even if the routine that called the privileged routine does not have permission to perform those same actions."

As noted above, reexamination takes a broad interpretation of claim language. The term 'routine' fairly includes within its scope such terms as function, operation, program, or method.

Fischer discloses aggregated PAI authority permissions on the call stack. The aggregated PAI may be effectively overridden by permissions granted by a "trusted authority." Fischer discloses (16: 13-65) that a program may have been signed with a public key or digital certificate, by the manufacturer, that grants a level of authority

Art Unit: 2173

(allowed to perform certain actions) to execute the program (routine). "The preferred methodology for determining whether the signatures are valid and whether they are trusted by the caller and whether the authority delegated by the program is permitted to have been delegated by the signer is taught in the inventor's U.S. Pat Nos. 4,868,877 and 5,005,200."

Fischer discloses (17: 67-18: 10), "...for called programs which carefully verify their own actions, it could be possible to allow the called program greater inherent authority (allow called program/routine to perform certain actions even if the program/routine that called the privileged routine does not have permission to perform those same actions) than the program which calls it...the possibilities for such combination must be carefully considered, not only by the designers of the underlying control system, but also by those who assign authority..." See FIG. 10, steps 306, 308, 320.

In this interpretation, a privileged called routine reads on a routine that is given greater inherent authority. The privileged called routine is referred to as a 'first routine', when the PCB representing a subsequent routine (a 'second routine') is placed on the call stack.

Fischer discloses **wherein the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with**

each routine in said calling hierarchy between and including said first routine and a second routine in said calling hierarchy, wherein said second routine is invoked after said first routine, wherein said second routine is a routine for performing said requested action.

Fischer discloses (10: 24-36) "an originating program" that "calls a program" having a program control block, or PCB. "Each new PCB will include a field... that points to the 'previous' of calling program control block." Then, "[w]hen a called program finishes executing, the system removes its associated PCB from the top of the executed stack, removes the associated program from storage, removes the associated authorizing information and accesses the program control block immediately below it in the stack." The call stack data structure grows from the bottom upwards. An 'originating program' places a PCB on the call stack and an additional PCB for each subsequent program added to the call stack. The PCB at the top of the call stack represents the "second routine" and the lower PCB on the call stack represents the "first routine", arranged as a calling hierarchy. Execution begins at the top of the stack. The "first routine" is "invoked" and an associated PCB is placed on the call stack. A call to a subsequent program/routine, upon authorization, will invoke a second routine, i.e., place PCB of second routine on call stack. See Fischer, FIG. 11, #354 & #358, second program completes its actions and PCB is removed from top of stack, effectively enabling thread of execution to execute next lower program (first routine) related to next lower PCB on stack. As discussed above (17: 47-61) permissions may be encompassed by permissions associated with each routine in said calling hierarchy (i.e. PAI permissions / constraints may be combined / aggregated).

Here the program further requests calls to other program and other functions of varying permission levels (see column 10, lines 7-52 and column 19, lines 5-15); and has associated with it a plurality of routines in a hierarchy, each level having an associated program control block (PCB) that links to parent or child levels with their own PCBs and lists program authorization information (PAI) for the particular sub-program being executed (see column 10, lines 23-57 and figure 5).

Also note, "The present invention, while it primarily focuses on defining functions which restrict the ability of a program to access resources normally allowed to users, could also, in an appropriate environment, be used to extend the capabilities beyond those normally allowed to a user. Thus, for example, programs whose PAI is signed by an authority recognized by the supervisor, could be allowed to perform extended functions." Fischer at 12:58-65.

Per claim 7, *Fischer* wherein the step of determining whether said permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first routine and said second routine further includes the steps of: determining whether said permission required is encompassed by at least one permission associated with said second routine.

Art Unit: 2173

Fischer discloses "an originating program" (a 'first routine' as it is placed in a lower position on the call stack) that "calls a program" (calling a second routine). Following a check of authorizations (if action to invoke second routine is authorized), the called routine is invoked, and the associated PCB is placed at top of the stack, resulting in a calling hierarchy represented by PCBs (first routine, second routine) on the call stack. As noted above (17: 40 – 18: 10), the authorizing information (constraints/permissions) found in the PAIs (associated with the PCBs and programs) is combined/merged as the stack grows. Due to the aggregated constraints (permissions), the 'routine' at the top of the call stack may encompass at least one permission associated with routines lower on the stack. See FIG. 10. (15: 63-67), A calling program calls program X for execution. The associated program control block (associated with the program X) will not be added to the top of the execution stack until it is determined that program X is permitted to be invoked and verification is successful completed. As an example, for a call to Program X (first routine calling second routine), steps from #306 to #326 determine whether permission required is encompassed by at least one permission associated with said calling routine (calling routine/ first routine determines whether it is permitted to invoke called routine / second routine).

See 17: 52-54, "...a calling program may not be permitted to exceed its assigned bounds by merely calling another program."

Alternately, consider Fischer teachings (16: 26-44) of determining the level of authority via digital certificates and a public key. USPN 5,005,200 (incorporated by reference) teaches a hierarchy of certificates. Fischer ('200, 30: 9-12), "The signatures and

Art Unit: 2173

certificates are then checked to ensure that they in fact are authorized as described above in conjunction with FIG. 7."

See FIG. 7 and related text at '200, 21: 45-21: 13. Fischer discloses ('200, 22: 17-19), "All certificates must be accompanied by signatures which are themselves authorized by antecedent certificates." '200, 22: 50-55, "...a check is made to determine that antecedent certificates grant sufficient authority to the sub certificate signers to permit them to validly sign the certificate...the trust value in the certificate must be consistent with the antecedent... (determining whether said permission required is encompassed by at least one permission associated with said second routine)."

Fischer discloses in response to determining said permission required is encompassed by at least one permission associated with said second routine, then performing the steps of:

A) selecting a next routine from said plurality of routines in said calling hierarchy;

B) if said permission required is not encompassed by at least one permission associated with said next routine, then transmitting a message, indicating that said permission required is not authorized;

C) repeating steps A and B until: said permission required is not authorized by at least one permission associated with said next routine, there are no more routines to select from said plurality of routines in said calling hierarchy, or determining that said next routine is said first routine.

Art Unit: 2173

Fischer '717 discloses (FIG. 10; 16: 55 – 18: 41) the sequence of operations of a supervisor program (executing supervisor routine by principal) for controlling the processing of a program being executed in accordance with its program authorization information. See Fischer '717, 16: 26-37 referencing USPN 5,005,200, describing the 'routines'/ verification operations iteratively called in a hierarchy. Fischer '717, 16: 40-44, "If the manufacturer's pedigree is not acceptable (permission is not encompassed / authorized), the routine branches...the execution of the program is suppressed..."

Fischer '717, 17: 23-28, "In the process of suppressing the execution of the program, an error code or message will be returned to the calling program... (transmitting a message, indicating that said permission required is not authorized)."

The term "routine" is not narrowly defined by the '476 specification and broadly reads on a program, routine, operation, procedure, function, etc., such as calling a program to invoke or iterating through certificates and signatures when analyzing a hierarchy of certificates as described by Fischer '200.

As an example, see USPN 5,005,200 to Fischer (incorporated by reference) FIG. 7.

See '200, 16: 59-63, "...recipient analyzes the certificates associated with the signature to determine that the proper authority has been conveyed to each certificate through its signatures and the antecedent certificate(s) of these authorizing signatures." '200, 22: 24-29, "The recipient examines every signature supplied and verifies that each

Art Unit: 2173

accurately signs its purported object...using the procedure detailed in FIG. 3. The recipient ensures that each signature includes a corresponding validated certificate (where the recipient is the calling program that uses various routines to verify permissions)." Fischer '200, 22: 46-56, "...a check is made to ensure that all certificates except the meta-certificate have at least one signature...a check is made to ensure that all necessary cosignatures for all presented objects are present...a check is made to determine that antecedent certificates grant sufficient authority to the sub certificate signers to permit them to validly sign the certificate...the trust value in the certificate must be consistent with the antecedent..." The certificates are nested, and a first routine checking a first certificate for permissions, is followed by a next routine checking a second certificate (a sub certificate) for permissions in the hierarchy of certificates. Fischer '200, 22: 19-21, The chain of antecedent certificates is traced back recursively to the meta-certificate (there are no more routines to select from said plurality of routines in said calling hierarchy).

Per **claim 10**, Fischer discloses a computer-readable medium version of the limitations addressed in claim 1 above. See 'Fischer 4: 63; '717 FIG. 3 and 7: 14-35,"...FIG. 3A ...exemplifies how program authorization information is stored...in association with a program...identifies the location on disk 98 of the associated program...an indicator 84, 90,...96...which identifies the location of its associated program authorization information, e.g., PAI 1....stored in a separate memory device 100...stored in the same memory media as its associated program (computer readable medium) ..."

Art Unit: 2173

Claims **11-16** are computer readable medium versions of claims 2-7 respectively. See limitations addressed in claims 2-7 and 10 above. Per **claim 15**, see "sequence of instructions" as shown in Fischer '717, FIGs. 10 & 11.

Claims **19-21** are system versions of claims 1, 3, and 4 respectively. See limitations addressed above. Fischer '717 teaches (Abstract; 4: 24-61) a system.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented 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 to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

Claims 1-7, 10-16, 19-21 are rejected under 35 U.S.C. 103(a) as being obvious over USPN 5,958,050 to Griffin in combination with Chan.

Regarding claims 8-9 and 17-18, there are no obvious teachings in Griffin or Chan suggesting setting a flag associated with said first routine to indicate that said first routine is privileged; and the step of determining that said next routine is said first routine includes determining that a flag associated with said next routine indicates said next routine is privileged.

Art Unit: 2173

Per **claim 1**, Griffin discloses (1: 22-25) **a method for providing security**, i.e., "for management of trust relationships among code segments to be executed inside a trust boundary."

Griffin discloses (9: 56-64) **detecting when a request for an action is made by a principal**, e.g., checking the relevant security provisions when it receives a request for "execution access for the class." "If it is determined that clearance to trust is required to grant a particular access, a path of trust must be found before the access will be granted by the trust manager. Clearance to trust might be required, for example, where a policy claim states that a particular class must be checked prior to being executed and the access requested in execution access for the class. This requirement is set by a MustCheckClaim policy claim." Griffin infers the 'principal' by ('050, 5: 22-24) disclosing a client computer 10...under the direction of...a program it is executing (execution process / thread- maps to claim term 'principal'). Griffin suggests ('050, 7: 44) the 'principal' may be executing a Web browser 16. See '050, 6: 33-51 & FIG. 3.

Griffin discloses **in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein said permissions are associated with said plurality of routines based on a first association between protection domains and permissions.**

"The trust manager system also includes a code examiner adapted to analyze a portion of code to determine potential resource use of the portion of code (detecting the

Art Unit: 2173

request) and a trust evaluator adapted to evaluate certificate requirements of the portion of code based on policy rules extracted from the policy file and the potential resource use specified by the code examiner (determining whether said action is authorized based on permissions associated with said principal). The trust evaluator also determines, from certificates from the certificate repository and a code identifier identifying the portion of code, whether execution of the portion of code is allowed by the policy rules given the potential resource use, the code supplier and applicable certificates." Griffin, 3:33-57. For example, "[i]n one embodiment of a trust manager according to the present invention, the trust manager examines each new class before it is allowed to load (association between protection domains and permissions), execute or otherwise gain control of resources by examining a set of claims (permissions) in a policy file and a certificate repository (associated protection domain)." Griffin, 3:34-38. "A certificate contains one or more claims, where a claim is a data structure defining a security policy of assertion about a class... (permissions)" Griffin, 3: 42-45. In response to detecting the request, determining whether said action is authorized by performed by a 'code examiner' and a 'trust evaluator.' Griffin, 3: 47-49. "When a new class is introduced to local computer 100, modifications to class loader 124 cause a trust manager 122 to be called before a new class is loaded. The code of the class is also provided to a code analyzer 120 (code examiner/ detecting the request) which determines what classes are called and what possible computer resources might be used by the code. With this information, the trust manager reads certificates...(permissions associated with a plurality of routines / reads and proves the

Art Unit: 2173

hierarchy of certificates / permissions recursively back to a 'policy claim', through plurality of routines / determines whether said action is authorized)." *Griffin*, 6: 52-57. The nested hierarchy of certificates required to be read and proven are reads on the "plurality of routines."

"The operation of the system shown in FIG. 3 will now be described with reference to the flowcharts of FIGS. 4 and 5. Once a class is obtained by local computer 100, usually in response to navigation to a site which sends out Java applets, the class loader attempts to load the class. Because of the hooks placed in the standard class loader 124 (trust management modifications to classloader.c in the standard Java runtime), the class loader 124 does not pass the class on to applet runtime executive 126 unless it receives an "OK-to-load" signal from trust manager 122. Code analyzer 120 determines a unique identifier for the class (S 1), as well as a superclass reference, a list of subclasses for the class, methods of the class and a hash code of the class. Once code analyzer 120 determines this information, it passes the information to trust manager 122." *Griffin*, 7:10-25.

See *Griffin*, FIGs. 4 & 5. "If, at step S6, the trust manager determines that a subclass or method needs to be checked, that is done (S7) by the process described in FIG. 5 as in step S3. If the trust manager finds that claim to load a class with the checked subclass or method (S8) or the trust manager determined in step S6 that checking was not required, the trust manager instructs the class loader (via an OK-to-load signal, or otherwise) to load the class (S9). If, at step S4 or step

Art Unit: 2173

S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S 10). At the conclusion of the process, the trust manager instructs the class loader to either load or not load the class. It should be understood that the trust manager's output is not limited to use in making a load/no load decision, but to the more general question of trust or no trust. For example, the trust manager's determination could be used to decide whether to execute code or not (in most cases, controlling code loading and code execution can achieve the same security goals), whether to forward a message or not, or to perform a security function or not." Griffin, 7:48-67.

This claim element would further have been obvious at the time of the invention to one of ordinary skill in the art from the teachings of Griffin, in combination with Chan. Chan notes that programs like web browsers typically define security managers. Chan discloses details of the JAVA class libraries and specifically notes "[a] security manager enforces security policies related to what a program is allowed to do." Chan, 1188. Chan deploys its security management system by, in part, referencing the source of the code currently being executed on the stack. In further detail: "A security manager enforces security policies related to what a program is allowed to do [A]pplications like Web browsers typically define a security manager and use System.setSecurityManager to install it..." Chan, 1188. For example, Chan discloses certain "Execution Stack Information," whereby "[t]he execution stack (call stack) is a record of the method calls that were made from the main program (principal thread / process calls made to first routine, second routine, etc.) to the current method." Chan, 1189. This execution stack "indicates all the methods that are in progress and pending termination of the current method

Art Unit: 2173

call." *Id.* More specifically, consider an exemplary embodiment disclosed by Chan: "For example, if main() calls foo(), which in turn calls bar(), the execution stack when executing inside bar() would be bar() -> foo() -> main(). For some methods to perform some of the permission checking (routines), they may need to inspect the execution stack to find out information about the current execution context (first association). The SecurityManager class provides protected methods that can be used by subclasses of the SecurityManager for this purpose." Chan, 1189. Thus it is clear that Chan, in response to detecting a request from a method currently on the execution stack, will determine whether the requested action is authorized based on the permissions associated with the cascading hierarchy of calls.

Therefore it would have been obvious, to one of ordinary skill in the art, at the time of the invention to modify Griffin's disclosure of trusted execution of a web browser, with the teachings of Chan that provide more text book detail regarding the routines of a security manager as defined for a web browser.

Per claim 2, Griffin discloses detecting when a request for an action is made by a thread; the step of determining whether said action is authorized includes determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said thread.

Art Unit: 2173

"The preferred embodiment for use with Java applets includes the class loader and runtime executive from the Java runtime system, modified according to the present invention."

For example, "[i]n one embodiment of a trust manager according to the present invention, the trust manager examines (trust manager process execution thread examines) each new class before it is allowed to load (load into calling hierarchy), execute or otherwise gain control of resources by examining a set of claims in a policy file and a certificate repository (examine each class's association between permissions and plurality of routines in calling hierarchy)." Griffin, 3: 34-38. "If it is determined that clearance to trust is required to grant a particular access, a path of trust must be found before the access will be granted by the trust manager. Clearance to trust might be required, for example, where a policy claim states that a particular class must be checked prior to being executed and the access requested in execution access for the class.

This requirement is set by a MustCheckClaim policy claim." Griffin, 9:56-63. "If, at step S6, the trust manager determines that a subclass or method needs to be checked, that is done (S7) by the process (process execution thread) described in FIG. 5 as in step S3... the trust manager's determination could be used to decide whether to execute code or not (in most cases, controlling code loading and code execution can achieve the same security goals)...." Griffin, 7:48-67.

This claim element would further have been obvious at the time of the invention to one of ordinary skill in the art from the teachings of Griffin in combination with Chan. Chan discloses that "[a] security manager enforces security policies related to what a program is allowed to do." Chan, 1188. "[A]pplications like Web browsers typically define a security manager ..." Chan, 1188. For example, Chan discloses certain "Execution Stack Information," whereby "[t]he execution stack is a record of the method calls that were made from the main program to the current method." Chan, 1189. This execution stack "indicates all the methods that are in progress and pending termination of the current method call." *Id.*

Per **claim 3**, Chan discloses (1189) **a calling hierarchy that includes a first routine**, as described in the hierarchy of calls described with reference to the execution stack.

Griffin discloses **the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with said first routine.**

"If it is determined that clearance to trust is required (whether said action is authorized) to grant a particular access, a path of trust must be found (determining whether a permission required to perform said action is encompassed) before the access will be granted by the trust manager."

Griffin, 9:56-63. "...Proving is done by finding a chain of claims from a claim about the class being loaded (check permissions associated with said first routine, where 'first routine' is

Art Unit: 2173

associated with routine loaded on execution stack) to a claim setting out a policy statement. A certificate contains one or more claims, where a claim is a data structure defining a security policy of assertion about a class, package of classes, or an entity to be trusted or not trusted.....The trust evaluator also determines, from certificates from the certificate repository and a code identifier identifying the portion of code, whether execution of the portion of code is allowed (determining whether said action is authorized) by the policy rules given the potential resource use, the code supplier and applicable certificates." Griffin, 3:33-57.

Per claim 4, Griffin discloses a method wherein the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy.

For example, "[i]n one embodiment of a trust manager according to the present invention, the trust manager examines each new class (determine authorization by examining each new class for permissions associated) before it is allowed to load (before adding routine to call stack / calling hierarchy), execute or otherwise gain control of resources by examining a set of claims in a policy file and a certificate repository." Griffin, 3:34-38. "The operation of the system shown in FIG. 3 will now be described with reference to the flowcharts of FIGS. 4 and 5. Once a class is obtained by local computer 100, usually in response to navigation to a site which sends out Java applets, the class loader attempts to load the class. Because of the hooks placed in the standard class loader 124 (trust management modifications to classloader.c in the standard Java

Art Unit: 2173

runtime), the class loader 124 does not pass the class on to applet runtime executive 126 unless it receives an "OK-to-load" signal from trust manager 122. Code analyzer 120 determines a unique identifier for the class (S 1), as well as a superclass reference, a list of subclasses for the class, methods of the class and a hash code of the class. Once code analyzer 120 determines this information, it passes the information to trust manager 122." Griffin, 7:10-25.

"If it is determined that clearance to trust is required to grant a particular access, a path of trust must be found before the access will be granted by the trust manager.

Clearance to trust might be required, for example, where a policy claim states that a particular class must be checked prior to being executed and the access requested in execution access for the class.

This requirement is set by a MustCheckClaim policy claim." Griffin, 9:56-63.

Per **claim 5**, Griffin discloses a **method for providing security**,

"The present invention relates to the field of trust management in a distributed control environment. More specifically, one embodiment of the invention provides for management of trust relationships among code segments to be executed inside a trust boundary." Griffin, 1: 21-25.

Griffin discloses **detecting when a request for an action is made by a principal**.

Griffin does this by, e.g., checking the relevant security provisions when it (when the executing process / principal makes a request) receives a request for "execution access for the class." Griffin, 9:61-62. "If it is determined that clearance to trust is required to

Art Unit: 2173

grant a particular access, a path of trust must be found before the access will be granted by the trust manager. Clearance to trust (by executing process / principal) might be required, for example, where a policy claim states that a particular class must be checked prior to being executed and the access requested in execution access for the class. This requirement is set by a MustCheckClaim policy claim." Griffin, 9:56-64. "... while the trust manager described above uses a trust decision to decide whether or not to load a class, in a variant of that trust manager, all classes might be loaded and the trust decision is used to decide whether or not to execute the code..." Griffin, 10:49-56. "Certificates are signed so that they are difficult to falsify. The trust manager system also includes a code examiner adapted to analyze a portion of code to determine potential resource use (detecting when a request for an action is made by a principal) of the portion of code..." Griffin, 3:33-57.

Griffin discloses **determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said principal.**

"The operation of the system shown in FIG. 3 will now be described with reference to the flowcharts of FIGS. 4 and 5. Once a class is obtained by local computer 100 (by executing process on computer, i.e., the principal), usually in response to navigation to a site which sends out Java applets, the class loader attempts to load the class. Because of the hooks placed in the standard class loader 124 (trust management

Art Unit: 2173

modifications to classloader.c in the standard Java runtime), the class loader 124 does not pass the class on to applet runtime executive 126 unless it receives an "OK-to-load" signal from trust manager 122 (if action is authorized). Code analyzer 120 determines a unique identifier for the class (S 1), as well as a superclass reference, a list of subclasses for the class, methods of the class and a hash code of the class. Once code analyzer 120 determines this information, it passes the information to trust manager 122." Griffin, 7:10-25. See Griffin FIGs 4 & 5. "... the trust manager instructs the class loader to either load (load onto calling hierarchy / call stack) or not load the class..." Griffin, 7:48-67. "In one embodiment of a trust manager according to the present invention, the trust manager examines each new class before it is allowed to load, execute or otherwise gain control of resources by examining a set of claims in a policy file and a certificate repository. The trust manager proves a claim before allowing a class to be loaded if a policy statement requires proof. Proving is done by finding a chain of claims from a claim about the class being loaded to a claim setting out a policy statement (association between permissions and a plurality of routines in a calling hierarchy associated with said principal). A certificate contains one or more claims, where a claim is a data structure defining a security policy of assertion about a class, package of classes, or an entity to be trusted or not trusted. Certificates are signed so that they are difficult to falsify. The trust manager system also includes a code examiner adapted to analyze a portion of code to determine potential resource use of the portion of code and a trust evaluator adapted to evaluate certificate requirements of the portion of code based on policy rules extracted

Art Unit: 2173

from the policy file and the potential resource use specified by the code examiner. The trust evaluator also determines, from certificates from the certificate repository and a code identifier identifying the portion of code, whether execution of the portion of code is allowed by the policy rules given the potential resource use, the code supplier and applicable certificates." Griffin, 3:33-57.

Chan provides supporting details regarding security manager functionality typically found in Web browsers (where web browsers were disclosed by Griffin). Chan discloses that "[a] security manager enforces security policies related to what a program is allowed to do." Chan, 1188. Chan deploys its security management system by, in part referencing the source of the code currently being executed on the stack. "A security manager enforces security policies (permissions) related to what a program is allowed to do [A]pplications like Web browsers typically define a security manager and use `System.setSecurityManager` to install it..." Chan, 1188. Chan discloses certain "Execution Stack Information," (calling hierarchy) whereby "[t]he execution stack is a record of the method calls that were made from the main program to the current method." Chan, 1189. This execution stack "indicates all the methods that are in progress and pending termination of the current method call." *Id.* For some methods to perform some of the permission checking, they may need to inspect the execution stack to find out information about the current execution context (calling hierarchy associated with said principal). The `SecurityManager` class provides protected

Art Unit: 2173

methods that can be used by subclasses of the SecurityManager for this purpose."

Chan, 1189.

Griffin discloses **each routine of said plurality of routines is associated with a class**. Griffin discloses (6: 48-51) class loaders from a Java runtime system. Such a class loader evaluates routines associated with a class.

Griffin discloses **said association between permissions and said plurality of routines is based on a second association between classes and protection domains**.

"If it is determined that clearance to trust is required to grant a particular access (association between permissions and plurality of routines), a path of trust must be found before the access will be granted by the trust manager. Clearance to trust might be required, for example, where a policy claim states that a particular class must be checked prior (second association between classes to be loaded and trust / protection domains) to being executed and the access requested in execution access for the class. This requirement is set by a MustCheckClaim policy claim."

Griffin, 9:56-63.

Per **claim 6**, Griffin discloses (1: 22-25) **a method for providing security**. "The present invention relates to the field of trust management in a distributed control environment. More specifically, one embodiment of the invention provides for management of trust relationships

Art Unit: 2173

among code segments to be executed inside a trust boundary." See Griffin method claims 1-12, 11:63-14: 13.

Griffin discloses **detecting when a request for an action is made by a principal.**

See limitations addressed in claim 1 above.

Griffin discloses **in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein a first routine in said calling hierarchy is privileged.**

Claim 6 differs from claim 1 with the recitation, **"wherein a first routine in said calling hierarchy is privileged."**

Gong '476 discusses the term 'privileged' at 3: 33-37; 6: 35-39; 13: 22-35: "...certain routines may be 'privileged'. A privileged routine is allowed to perform certain actions even if the routine that called the privileged routine does not have permission to perform those same actions." "When determining whether a thread is able to perform an action, only the permissions associated with the privileged routine and the routines above the privileged routine in the calling hierarchy of the thread are inspected." "...a method may cause itself to be privileged (i.e. enable

Art Unit: 2173

the privilege mechanism) by invoking a method of a privilege object called, for example, beginPrivilege.”

Broadly Gong '476 discloses (3: 3-11) the term 'routine' includes within its scope such terms as functions or methods. Actions are authorized based on permissions associated with calling routines (authorized called routines invoked and placed on the call stack). “A calling hierarchy indicates the routines (e.g. functions, methods) that have been invoked (and placed on the call stack) by or on behalf of a principal (e.g. thread, process) but have not been exited.”

Broadly Griffin discloses (4: 49-57) that “[t]he process of proving a claim is one of verifying the claimant, a level of trust in the claimant and the authorization of that claimant (prove by verifying claim is privileged), typically using other claims. This leads to a chain of claims ending with a known trust claim or claimant. A claim makes an assertion. An assertion could be that a particular element can be trusted, that a particular class should not be trusted, that a particular class can be trusted, etc.”

Griffin in view of Chan discloses the step of determining whether said action is authorized further includes **determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first routine and a**

Art Unit: 2173

second routine in said calling hierarchy, wherein said second routine is invoked after said first routine, wherein said second routine is a routine for performing said requested action.

A calling hierarchy represents the sequence of calls to routines (calls to programs / procedures / functions/ methods / operations, etc), where a requested action (by calling routine) to load the routine (action requesting to invoke/load called routine) on the call stack (an action to invoke a routine) is made by the principal thread or process of execution. If each routine requested to be loaded is authorized to be loaded, it is sequentially placed on the call stack / execution stack (invoked). The routine associated with the top of the call stack represents the 'second routine.' The routine associated with the position just prior to the second routine represents the 'first routine.'

Chan discloses that "[a] security manager enforces security policies related to what a program is allowed to do." Chan, 1188. Chan deploys its security management system by, in part, referencing the source of the code currently being executed on the stack. In further detail: "A security manager enforces security policies related to what a program is allowed to do [A]pplications like Web browsers typically define a security manager and use System.setSecurityManager to install it..." Chan, 1188. Chan discloses certain "Execution Stack Information," whereby "[t]he execution stack is a record of the method calls (calls to routines) that were made from the main program (from the principal) to the current method." Chan, 1189. This execution stack "indicates all the methods that are in progress (each routine in

a calling hierarchy) and pending termination of the current method call." *Id.* (where current method call references the top of the stack or otherwise the 'second routine; first routine is referenced after the second routine exits execution).'

"... For some methods to perform some of the permission checking, they may need to inspect the execution stack to find out information about the current execution context. The SecurityManager class provides protected methods that can be used by subclasses of the SecurityManager for this purpose." Chan, 1189. Thus it is clear that Chan, in response to detecting a request from a method currently on the execution stack, will determine whether the requested action is authorized based on the permissions associated with the cascading hierarchy of calls (permissions associated with each routine in said calling hierarchy / permissions of routines prior to the 'second routine', i.e. permissions of the first routine).

Per **claim 7**, Griffin in view of Chan discloses **the step of determining whether said permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first routine and said second routine further includes the steps of: determining whether said permission required is encompassed by at least one permission associated with said second routine; and in response to determining said permission required is encompassed by at least one permission associated with said second routine, then performing the steps of:**

Art Unit: 2173

A) selecting a next routine from said plurality of routines in said calling hierarchy, B) if said permission required is not encompassed by at least one permission associated with said next routine, then transmitting a message indicating that said permission required is not authorized, and C) repeating steps A and B until: said permission required is not authorized by at least one permission associated with said next routine, there are no more routines to select from said plurality of routines in said calling hierarchy, or determining that said next routine is said first routine.

Griffin discloses (9: 12-16) that "[t]he trust manager will keep searching for a path through the web of claims, using hints as needed, moving up and down along a path (repeat with next routine) as dead ends are encountered. Eventually, the trust manager may exhaust all possible hints and have searched all available claims. "If it is determined that clearance to trust is required to grant a particular access, a path of trust must be found before the access will be granted by the trust manager.... " Griffin, 9:56-64. Griffin discloses (13: 17-18),"...when a path of trust cannot be found after a finite search, denying the object the access."

Chan discloses that "[a] security manager enforces security policies related to what a program is allowed to do." Chan, 1188. Chan deploys its security management system by, in part, referencing the source of the code currently being executed on the stack. In further detail: "A security manager enforces security policies related to what a program

Art Unit: 2173

is allowed to do[A]pplications like Web browsers typically define a security manager... " Chan, 1188.

Chan discloses certain "Execution Stack Information," whereby "[t]he execution stack is a record of the method calls that were made from the main program to the current method." Chan, 1189. This execution stack "indicates all the methods that are in progress and pending termination of the current method call." *Id.*

Chan discloses (Chan, 1189) a recursive inspection of the call stack, until there are no more routines to select from: "For example, if main() calls foo(), which in turn calls bar(), the execution stack when executing inside bar() would be bar() -> foo() -> main(). For some methods to perform some of the permission checking, they may need to inspect the execution stack to find out information about the current execution context..."

Griffin discloses "denying the object the access" when a path of trust cannot be found.

Griffin fails to disclose transmitting a message indicating that said required permission is not authorized. It would be obvious to transmit a message indicating that said required 'permission' is not authorized. See Griffin, FIG. 5, last step. To transmit such a message would be within the skill level of one of ordinary skill in the art and would provide useful information to a program operator."

Art Unit: 2173

Chan shows that methods of the Security Manager can throw an exception. It would be obvious to one of ordinary skill in the art to create within the exception code block a message to be transmitted indicating that said permission required is not authorized. Such a modification would be well within the skills of one of ordinary skill in the art.

Claim 10 is a computer readable medium version of limitations addressed in claim 1 above.

Griffin discloses (FIG. 2) a computer, including a CPU and hard drive and (FIG. 3) the trust manager software components stored on a client computer being protected. See Griffin (7: 15-17) “trust management modifications to classloader.c in the standard Java runtime” in reference to code modifications stored on a computer readable medium of a trusted delegation system.

See limitations of **claims 11-16** addressed above in claims 2-7 respectively.

Claim 19 is a system version of limitations addressed in claim 1 above. Griffin discloses (3: 47; 6: 33-34; 14: 14-60) a system that performs the methods steps previously discussed. “FIG. 3 is a detailed block diagram of a trust management system which can provide the access control for a local computer 100.”

Claims 20 and 21 are system versions of limitations addressed in claims 3 and 4 above.

Additional teachings by Organick are cumulative to the art. Proposed rejections in view of Organick are held in abeyance.

In summary, claims 1-7, 10-16, and 19-21 are rejected. Claims 8- 9 and 17-18 are confirmed, as there are no prior art disclosures of the "setting a flag" limitation.

Response to Arguments

This is a response to the request for reconsideration filed 9-16-2011, in response to the non-final action mailed 6-16-2011.

Arguments against the 35 U.S.C. 102(b) rejections over Fischer:

IV.

A.

1.

The Patent Owner argues (from page 23) that Fischer's System Monitor does not meet the claimed requesting principal.

As directed toward by the patent owner, in the interview Agenda of 8-12-2011, the "principal" is defined in the '476 specification (2: 35-38) as "an entity in the computer system to which permissions are granted" (e.g., processes, objects, and threads). In

Fischer the "system monitor" (or alternately the program) is being compared to the claimed "principal", here the system monitor is a software component that "builds" a data structure including a set of authorities, referred to as "program authorization information" or PAI (see column 2, lines 16-27). The system monitor further utilizes a "program control block" or PCB to control execution of the program (see column 10, lines 8-54), where the PCB picks through a stack of associated programs to determine authorizations granted at each level.

The Patent Owner argues that Fischer does not disclose "detecting when a request for an action is made by a principal", because the system monitor does not request an action.

The Examiner respectfully submits that the system monitor utilizes the PCB to "control the execution of an associated program", where the PCB is further loaded with program authorization information (PAI) referenced to make sure that the associated program has the sufficient authorizations (see column 10, lines 8-54). So the system monitor, in conjunction with the PCB and PAI request execution of a program, request a checking of associated permission, and request transfer of execution between hierarchically arranged program blocks.

The Patent Owner argues that if the system monitor is identified as the "principal", then Fischer does not disclose "a plurality of routines in a calling hierarchy

Art Unit: 2173

associated with said principal", because the programs are not associated with the system monitor.

The Examiner respectfully submits that the system monitor limits the ability of a program about to execute, by providing linkages to associated PCB and PAI information (see column 2, lines 16-27). The system monitor utilizes a program control block (PCB) for each level of a plurality of levels of a program (see column 9, line 64 through column 10, line 53).

2.

The Patent Owner argues (from page 25) that Fischer's "Program" does not meet the claimed requesting principle, as if Fischer's program is identified as the "principal," then Fischer does not disclose "permissions associated with a plurality of routines in a calling hierarchy associated with said principal".

The Examiner respectfully submits that Fischer's program can also be read as the principle. Again the claimed "principle" is defined as: from the '476 specification (2: 35-38) as (1) "an entity in the computer system to which permissions are granted" (e.g., processes, objects, and threads), (2) where the principle makes "a request for an action", and (3) where "a plurality of routines in a calling hierarchy associated with said principal". The Examiner will now address how each of these characterizations of the "principle" are met by Fischer's program:

(1) The program is clearly a process (executable set of code) ran by the computer system (see column 9, line 19 through column 10, line 7) and requesting

Art Unit: 2173

usage of the computers resources (see column 10, lines 7-52 and column 2, lines 34-49).

(2) The program further requests calls to other programs and other functions of varying permission levels (see column 10, lines 7-52 and column 19, lines 5-15).

(3) The program further has associated with it a plurality of routines in a hierarchy, each level having an associated program control block (PCB) that links to parent or child levels with their own PCBs and lists program authorization information (PAI) for the particular sub-program being executed (see column 10, lines 23-57 and figure 5).

B.

1.

The broad meaning of the Patent Owner's "routine" is pointed toward on page 26 of the response stating that:

"Further, according to Professor Goldberg's declaration, 'a program in an object-oriented language (e.g. Java and C++) may include routines. In the context of the '476 Patent, one of ordinary skill would understand that a routine is a portion of code (i.e. a set of executable instructions or statements) within a program that may be called from other routines - hence the existence of a 'calling hierarchy' of routines. For example, as noted in the '476 Patent, a routine may be a function or method ('476, 2:38).' (Goldberg Declaration, paragraph 17.) Thus, Fischer's programs do not disclose 'a plurality of routines in a calling hierarchy' as expressly required by the claims and therefore the

Art Unit: 2173

PAIs' assignment to the program does not meet "permissions associated with a plurality of routines in a calling hierarchy."

The Patent Owner argues that Fischer's PAI is associated with a program and thus cannot meet "permissions associated with a plurality of routines in a calling hierarchy.

The Examiner respectfully submits that Fischer's program has a plurality of sub-programs each with their own PCBs that define permissions for the program at various hierarchically arranged levels, where each of these sub-programs or routines, called by the parent program, have their own PAIs (program authorization information) (see column 10, lines 23-57 and figure 5).

The Patent Owner argues (from page 26) that program are not routines... and "determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principle".

The Examiner respectfully submits that in Fischer the parent program is the principle, and the parent program comprises a plurality of sub-programs or routines. Each of these hierarchically arranged routines have an associated program control block (PCB) that defines the individual permissions (via PAI) for the sub-program routine (see column 10, lines 23-57 and figure 5).

2.

The Patent Owner argues (from page 27) that the "hierarchy of nested certificates and signatures" is not a plurality of routines in a calling hierarchy.

The Examiner respectfully submits that the certificates and signatures are not being treated as the routines, the routines are the sub-program elements of the parent program, that each have associated sub-program control blocks (PCBs) that contain the certificates and signatures (PAI), used to determine permissions at each level.

The Patent Owner argues that although these routines may check certificates associated with a PAI, Fischer does not disclose determining whether an action is permitted based on permissions associated with these checking routines.

The Examiner respectfully submits that Fischer teaches, in column 2, lines 16-36, the "PAI defines the range of operations that a program may execute and/or defines those operations that a program cannot perform."

C.

The Patent Owner argues (from page 28) that Fischer does not disclose the claimed "each routine of said plurality of routines is associated with a class" and "a second association between classes and protection domains".

The Examiner respectfully submits that Fischer teaches, in column 3, lines 11-20 and column 7, line 29 through column 8, line 49, a system in which object oriented

Art Unit: 2173

programming is used for custom programs using a generic parent class. Here a parent originating program, defines a protection domain for the parent class, sub-class routines have their own respective protection domains.

The Patent Owner argues (from page 29) that that Fischer does not disclose the claimed "a second association between classes and protection domains".

The Examiner respectfully submits that Fischer teaches, in column 3, lines 11-20, if a parent program / class task is not permissible execution of child programs / classes will be suppressed, yet successful confirmation of security / authority of a parent program still may require confirmation of a child programs security.

The Patent Owner argues, from paragraphs 21 and 22 of the Goldberg declaration that every portion of the program in Fischer has the same permissions, and thus Fischer does not provide a mechanism for associating different permission with different portions of code.

The Examiner respectfully submits that Fischer teaches, in column 10, lines 8-57, different program control blocks (PCBs) associated with sub-programs (routines), each having their own assigned program authorization information (PAI) defining the set of permissions for that particular program segment, thereby allows different subsets of code to have different associated permissions.

D.

The Patent Owner argues (on page 32) that Fischer doesn't teach a first routine in said calling hierarchy being privileged and further determining a permission associated with routine in the calling hierarchy and further argues that "unlike Fischer, the claimed first, privileged routine is not performing the requested action: the subsequently called second routine is performing the action" (from paragraph 24 of the Goldberg declaration).

The Examiner respectfully submits that in Fischer a parent routine can be privileged (having access granted in the PAI), as the program executes however, sub-programs (routines) that are executed have their program control block (PCB) checked to assure that their permissions (PAI) are being watched as well, these sub permissions defining "a range of operations that a program may execute and / or defines those operations that a program cannot perform" (see column 2, lines 34-48 and column 10, lines 8-57). These programs are not checked until they are called as can be seen from the flow diagram in figure 5.

E.

The Patent Owner only argues dependent claims "for at least the same reasons as their respective independent claims."

F.

Art Unit: 2173

The Patent Owner argues against the order being granted as the "current rejection resurrects a nearly identical rejection made and overcome in the original examination".

The Examiner respectfully submits that as pointed out in the order the hierarchy of nested permissions is being viewed in a new light, considering the program control blocks (PCB) and associated program authorization information (PAI) to be a hierarchy of sub-programs each with its own assigned authorization.

V.

Arguments against the 35 U.S.C. 103(a) rejections over Griffin in combination with Chan:

A.

1.

The Patent Owner argues that neither Griffin nor Chan teach or suggest a "principal" that requests an action, where the same "principal" is associated with "a plurality of routines in a calling hierarchy," and those routines are associated with "permissions."

The Examiner respectfully submits that Griffin's program can be read as the "principle". Again the claimed "principle" is defined as: from the '476 specification (2: 35-38) as (1) "an entity in the computer system to which permissions are granted" (e.g., processes, objects, and threads), (2) where the principle makes "a request for an

Art Unit: 2173

action", and (3) where "a plurality of routines in a calling hierarchy associated with said principal". The Examiner will now address how each of these characterizations of the "principle" are met by Fischer's program:

(1) The program is clearly a process (executable set of code) ran by the computer system (see column 3, lines 46-57) where the program is requesting usage of the computers resources (see column 3, lines 46-57, column 6, lines 37-44, and column 7, lines 15-25).

(2) The program further requests calls to other sub-programs (routines) of varying permission levels (see column 7, lines 15-25).

(3) The program further has associated with it a plurality of routines in a hierarchy, each level having an associated class that need be examined to determine potential resource use of the portion of code for the particular sub-program being executed (see column 3, lines 33-57 and in figure 5).

The Patent Owner argues (from page 37) that "as Professor Goldberg explains, however, 'this 'principal' is not associated with 'a plurality of routines in a calling hierarchy' that are associated with 'permissions.'" (Goldberg Declaration, paragraph 29.) "On the contrary," Professor Goldberg continues, "Griffin's 'trust manager instructs the class loader to either load or not load the class,' or to grant 'trust or no trust,' or 'execute code or not.'" (Goldberg Declaration, paragraph 29 citing Griffin at 7: 58-64.)"

The Examiner respectfully submits that the parent program (principle) is associated with a plurality of sub-class calls (routines) that need be evaluated for their

Art Unit: 2173

associated permissions (see column 7, lines 14-25). Just because a parent calling program is granted full system rights doesn't mean a program that is calls should have the same rights, each of the program segments (portions of code) of Griffin can be evaluated for authority. Alternately, a "claimant" may be used to define a chain of claims with a known trust level.

The Patent Owner argues (from page 38) against an alternate mapping of the principle to the Griffin reference's "trust manager".

The Examiner respectfully submits that though the mapping to the program appears to be a clearer mapping, a mapping to the "trust manager" still seems appropriate, as trust manager is a code segment that requests consideration of the access rights to each of the program elements and sub-program elements (see column 3, lines 32-57). With regard to an "association" with a plurality of routines in a hierarchy, the trust manager is the program element that provides checks of potentially every program and sub-program that is called to maintain security in the delegation system (see column 6, lines 33-47).

The Patent Owner argues (from page 39) that there is no teaching in Griffin or Chan that suggests determining authorization by checking permissions associated with the disclosed "methods that are in progress."

The Examiner respectfully submits that the claim is not so limiting to only check for the permissions of a currently executing section of code, this would appear

Art Unit: 2173

counterproductive to check the security of a code segment while executing and providing permissions to the program element. Griffin specifically teaches, in column 3, lines 32-57, making a determination of whether execution of the portion of code is allowed by the policy rules. Where a trust manager examines each new class before it is allowed to load. This however does not preclude a program from running yet checking upon call to sub-program / sub-class / sub-routine for associated sub-permissions (see column 7, lines 10-67). Chan further teaches a security manager that evaluates class / subclass permissions about the "current execution context" (see pages 1188 and 1189).

2.

The Patent Owner argues that the claimed "The Combination Does Not Disclose the Claimed "permissions associated with a plurality of routines in a calling hierarchy" (Claims 1-7, 10-16, 19- 21)".

The Examiner respectfully submits that Griffin teaches that the program further has associated with it a plurality of routines in a hierarchy, each level having an associated class that need be examined to determine a level of trust for the portion of code and potential resource use of the portion of code, in the particular sub-program being executed (see column 3, lines 33-57 and in figure 5).

3.

The Patent Owner argues (from page 41) that "The Combination Does Not Disclose the Claimed "plurality of routines in a calling hierarchy associated with [the] principal" (Claims 1-7, 10-16, 19-21)".

The Examiner respectfully submits that the plurality of sub-program / sub-class / sub-routine, are each "associated" with the parent calling program that initiated their execution (see column 7, lines 10-67).

The Patent Owner argues (from page 42) that moreover, alternative mappings for the required "plurality of routines in a calling hierarchy associated with said principal" similarly fail to disclose the claimed elements. For example, as Professor Goldberg indicates, "Griffin's trust manager routines that evaluate new classes against certificates fail to disclose the required 'plurality of routines' at least because Griffin' s certificates are not associated with the trust manager routines, but rather with the new class to be loaded." (Goldberg Declaration, paragraph 37.)

The Examiner respectfully submits that if the "trust manager" were interpreted to be the principle, the trust manager shows and association with the sub-programs / sub-classes / sub-routines, as it is the trust manager that decides whether each is executed or precluded from execution and / or limited as to which system resources it may access (see column 6, lines 33-51). Here the trust manager examines each new class / sub-class before allowing it to use system resources by examining the set of claims the program makes as well as certificates of authentication (see column 3, lines 32-56).

4.

The Patent Owner argues that "The Combination Does Not Disclose the Claimed "each routine..., is associated with a class" and "a second association between classes and protection domains" (Claims 5, 14)".

The Examiner respectfully submits Griffin teaches a principle that is associated with a class that is evaluated to determine the permission for the "principle" program, and that a plurality of sub-programs / sub-classes / sub-routines fall under the "principle" program in a hierarchy, each level having an associated class that need be examined to determine potential resource use of the portion of code, for the particular sub-program being executed (see column 3, lines 33-57 and in figure 5).

5.

The Patent Owner argues (from page 44) that "The Combination Does Not Disclose the Claimed "first routine in said calling hierarchy is privileged" (Claims 6, 7, 15, 16)."

The Examiner respectfully submits that the specifics of the privileged routine being argued are not claimed. More specifically, as stated from representative Claim 6, the limitation argued is:

"

wherein a first routine in said calling hierarchy is privileged

"

Since the interpretation of the limitation is the basis for the arguments, the Examiner's interpretation is now given. The claim, as interpreted by the examiner, pertains to a system in which certain routines are granted privileges to certain computer resources, where a routine that has been granted authorization is deemed "privileged". As stated in the eighth paragraph of MPEP 2101[R2].II.C.,

"Office personnel are to give claims their broadest reasonable interpretation in light of the supporting disclosure. In re Morris, 127 F.3d 1048, 1054-55, 44 USPQ2d 1023,1027-28 (Fed. Cir. 1997)."

With regard to the argument, Griffin teaches, a global proving that sub-classes are acceptable, via verifying a "claimant" (tree of trusted claims) (see column 4, lines 39-54). Thereby, if the claimant can be trusted then the claim as a whole, with all it's sub-structure, will be assumed to be authenticated / true.

6.

The Patent Owner argues that "The Combination Does Not Disclose the Claimed "permission associated with each routine in said calling hierarchy" (Claims 6, 7, 15, 16)".

The Examiner respectfully submits that Griffin teaches that the program further has associated with it a plurality of routines in a hierarchy, each level having an associated class that need be examined to determine potential resource use of the

Art Unit: 2173

portion of code, for the particular sub-program being executed (see column 3, lines 33-57 and in figure 5).

B.

1.

The Patent Owner argues that the combination changes the principle of operation of Griffin.

The Examiner respectfully submits that the addition of Chan merely defines a use of Class Libraries usable via the Java applets and associated "class loader" used with in the Griffin reference (see column 6, lines 33-51) and defines details regarding how a security manager evaluates class / subclass permissions about the "current execution context" (see pages 1188 and 1189). Where clarification of permission checking during the operation of a class / sub-class structure in a program does not change the operation of either reference. Here Griffin's trust manager examines each new class before it is allowed to load but may not even be called till mid-execution of the program. This does not preclude a program from running yet checking upon call to sub-program / sub-class / sub-routine for associated sub-permissions (see column 7, lines 10-67).

2.

The Patent Owner argues that the combination of Griffin and Chan is based upon impermissible hindsight reasoning.

Art Unit: 2173

In response to applicant's argument that the examiner's conclusion of obviousness is based upon improper hindsight reasoning, it must be recognized that any judgment on obviousness is in a sense necessarily a reconstruction based upon hindsight reasoning. But so long as it takes into account only knowledge which was within the level of ordinary skill at the time the claimed invention was made, and does not include knowledge gleaned only from the applicant's disclosure, such a reconstruction is proper. See *In re McLaughlin*, 443 F.2d 1392, 170 USPQ 209 (CCPA 1971). Griffin's inclusion of Java Class structures would have lead one of ordinary skill in the art to a guide for "Java Class Library", and considered the resultant combination / definition a state of the art at the time.

3.

The Patent Owner argues that the Requestor's reasons to combine are insufficient.

The Examiner respectfully submits that a prima facie case of obviousness has been shown through the 6-16-2011 office action stating that *"it would have been obvious, to one of ordinary skill in the art, at the time of the invention to modify Griffin's disclosure of trusted execution of a web browser, with the teachings of Chan that provide more text book detail regarding the routines of a security manager as defined for a web browser."* A guide (Chan) describing the operation of a class structure of specific software used in the other reference (Griffin) merely incorporates known structure of the specified software programming language.

C.

The Patent Owner argues (from page 49) that "Griffin fails to disclose determining authorization for an action requested by a thread based on permissions associated with the thread's routines." (Goldberg Declaration, paragraph 50.) "On the contrary, Griffin discloses determining the trustworthiness of a new class before it is loaded or executed." (*Id.*, paragraph 50) "Chan also does not disclose associating routines and permissions in the manner required." (Goldberg Declaration, paragraph 51.) Accordingly, claims 2, 3, 11, 12, and 20 are patentable over the combination of Griffin and Chan.

The Examiner respectfully submits that the claim is not so limiting to only check for the permissions of a currently executing section of code, this would appear counterproductive to check the security of a code segment while executing and providing permissions to the program element. Griffin specifically teaches, in column 3, lines 32-57, making a determination of whether execution of the portion of code is allowed by the policy rules. Where a trust manager examines each new class before it is allowed to load. This however does not preclude a program from running yet checking upon call to sub-program / sub-class / sub-routine for associated sub-permissions (see column 7, lines 10-67). Chan further teaches a security manager that evaluates class / subclass permissions about the "current execution context" (see pages 1188 and 1189).

The Patent Owner argues that "Griffin and Chan fail to disclose 'a first routine' with 'a permission required to perform said action [being] encompassed by at least one permission associated with said first routine,'" as required by these claims. (Id., paragraph 51.) "In contrast to this routine/permission association," Professor Goldberg points out that "Griffin 'examines each new class before it is allowed to execute' to determine trustworthiness, and consequently does not render obvious the recited features of claims 3, 12, and 20." (Goldberg Declaration at paragraph 51 citing Griffin at Abstract.) "Chan also does not disclose associating routines and permissions in the manner required." (Goldberg Declaration, paragraph 51.) Accordingly, claims 2, 3, 11, 12, and 20 are patentable over the combination of Griffin and Chan.

The Examiner respectfully submits that The Examiner respectfully submits that Griffin teaches that each of a plurality of sub-programs / sub-classes / sub-routines in a hierarchy has an associated class that need be examined to determine a permission for the code segment (see column 3, lines 33-57 and in figure 5). Where "...Proving is done by finding a chain of claims from a claim about the class being loaded (check permissions associated with said first routine, where 'first routine' is associated with routine loaded on execution stack) to a claim setting out a policy statement. A certificate contains one or more claims, where a claim is a data structure defining a security policy of assertion about a class, package of classes, or an entity to be trusted or not trusted The trust evaluator also determines, from certificates from the certificate repository and a code identifier identifying the portion of code, whether execution of the portion of code is allowed (determining whether said action is authorized) by the policy rules given

Art Unit: 2173

the potential resource use, the code supplier and applicable certificates." Griffin, 3:33-57.

The Patent Owner argues that Griffin's teaching to determine the trustworthiness of a class by checking potentially hierarchical certificates fails to teach or suggest the recited "at least one permission associated with each routine in [a] calling hierarchy," as required by claims 4, 13, and 21 in the '476 Patent.

The Examiner respectfully submits that Griffin teaches a principle that is associated with a class that is evaluated to determine the permission for the "principle" program, and that a plurality of sub-programs / sub-classes / sub-routines fall under the "principle" program in a hierarchy, each level having an associated class that need be examined to determine potential resource use of the portion of code, for the particular sub-program being executed (see column 3, lines 33-57 and in figure 5).

The Patent Owner argues that "Claims 7 and 16 are also patentable over the cited combination," according to Professor Goldberg, as "Griffin would not transmit a message indicating the 'permission required' for a routine in a calling hierarchy is not authorized because Griffin does not associate permissions with routines in a calling hierarchy." (Goldberg Declaration, paragraph 53.)

The Examiner respectfully submits that Griffin teaches that each of a plurality of sub-programs / sub-classes / sub-routines in a hierarchy has an associated class that need be examined to determine a trust level (see column 3, lines 33-57 and in figure 5).

VI.

A.

The Patent Owner argues that "the claimed invention satisfied a long-felt need" and further argues against the prior art approach of using a "sand box".

The Examiner respectfully submits that Patent Owner appears to show that the prior art is nothing more than a "sand box" approach to Java security. The references however, provide advanced layered authentication schemes that provide for selective access outside of the "box", while containing routines with security concerns within the "box" (see column 1, line 60 through column 2, line 48 of Fischer).

B.

The Patent Owner argues that "the claimed invention led to commercial success".

The Examiner respectfully submits that the evidence of commercial success is not convincing as there is no one to one mapping between the claims and what is taught in Exhibit F. Therefore it cannot be seen that the Oracle implemented security model was the same Patent Owners claimed work. The mere fact that a product using the general idea of the claimed invention has had commercial success does not necessarily make the claimed invention novel.

To be given substantial weight in the determination of obviousness or nonobviousness, evidence of secondary considerations must be relevant to the subject

Art Unit: 2173

matter as claimed, in this case the Examiner does not see a nexus between the merits of the claimed invention and the evidence of secondary considerations.

C.

The Patent Owner argues that "the claimed invention received industry praise soon after its introduction".

The Examiner respectfully submits that the Exhibits J, K, and L show an improved sandbox approach with added class / sub-class security policies, where the references applied supra are directed at the same or similar improvements, with an earlier filing date.

D.

The Patent Owner argues that "the claimed invention was copied by Google".

The Examiner respectfully submits that this is the goal of this reexamination to prove or disprove this point, in view of the substantial new question of patentability of the claimed invention.

Litigation Reminder

The patent Owner is reminded of the continuing responsibility under 37 CFR 1.565(a) to apprise the Office of any litigation activity, or other prior or concurrent proceeding, involving Patent Number: 5,694,322 throughout the course of this

Art Unit: 2173

reexamination proceeding. The third part requester is also reminded of the ability to similarly apprise the Office of any such activity or proceeding throughout the course of this reexamination proceeding. See MPEP §§ 2207, 2282 and 2286.

Conclusion

THIS ACTION IS MADE FINAL.

A shortened statutory period for response to this action is set to expire 2 months from the mailing date of this action.

Extensions of time under 37 CFR 1.136(a) do not apply in reexamination proceedings. The provisions of 37 CFR 1.136 apply only to "an applicant" and not to parties in a reexamination proceeding. Further, in 35 U.S.C. 305 and in 37 CFR 1.550(a), it is required that reexamination proceedings "will be conducted with special dispatch within the Office."

Extensions of time in reexamination proceedings are provided for in 37 CFR 1.550(c). A request for extension of time must be filed on or before the day on which a response to this action is due, and it must be accompanied by the petition fee set forth in 37 CFR 1.17(g). The mere filing of a request will not effect any extension of time. An extension of time will be granted only for sufficient cause, and for a reasonable time specified.

The filing of a timely first response to this final rejection will be construed as including a request to extend the shortened statutory period for an additional month, which will be granted even if previous extensions have been granted. In no event

Art Unit: 2173

however, will the statutory period for response expire later than SIX MONTHS from the mailing date of the final action. See MPEP § 2265.

All correspondence relating to this *ex parte* reexamination proceeding should be directed:

By Mail to: Mail Stop Ex Parte Reexam
Central Reexamination Unit
Commissioner for Patents
United States Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

By FAX to: (571) 273-9900
Central Reexamination Unit

By hand: Customer Service Window
Randolph Building
401 Dulany Street
Alexandria, VA 22314

Registered users of EFS-Web may alternatively submit such correspondence via the electronic filing system EFS-Web, at

<https://sp0rtal.uspto.gov/authenticate/authenticateuserlocalepf.html>.

EFS-Web offers the benefit of quick submission to the particular area of the Office that needs to act on the correspondence. Also, EFS-Web submissions are "soft scanned" (i.e., electronically uploaded) directly into the official file for the reexamination proceeding, which offers parties the opportunity to review the content of their submissions after the "soft scanning" process is complete. Any inquiry concerning this communication should be directed to the Central Reexamination Unit at telephone number (571) 272-7705.

/Dennis G. Bonshock/

Primary Examiner, Art Unit 2173

Central Reexamination Unit 3992

ALB
ARL




Approved for use through 07/31/2012 (MPEP 0081-0031)
 U.S. Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE
 Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

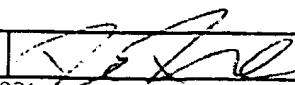
37 C.F.R. 1.501 INFORMATION DISCLOSURE CITATION IN A PATENT			Complete If Known		
			Docket Number	Patent Number	
			154892800500	6,192,476	
			Applicant		
			LI GONG		
Issue Date			Art Unit		
February 20, 2001			3992		
Sheet	1	of	3		

U.S. PATENT DOCUMENTS							
Examiner Initials*	Cite No.	Document Number Number, Kind Code* (if known)	Publication Date MM-DD-YYYY	Name	Class	Subclass	Page Date If Applicable

FOREIGN PATENT DOCUMENTS								
Examiner Initials*	Cite No.	Document Number	Date	Country	Class	Subclass	Translation	
							Yes	No

*EXAMINER: Initial information considered, whether or not citation is in conformance with MPEP 2614. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to the third party requester. Third party requester is to place a check mark here if English language translation is attached.

NON PATENT LITERATURE DOCUMENTS		
Examiner Initials*	Cite No.	Include name of the author (in CAPITAL LETTERS); title of the article (when appropriate); title of the periodical, magazine, journal, serial, symposium, catalog, etc.; date, origin; volume, issue, number(s); publisher; city and/or country where published.
  	1.	DEVELOPER.ANDROID.COM (August 24, 2011). "Android Developers: Package, Java Security," located at <http://developer.android.com/reference/java/security/package-summary.html>, last visited on August 27, 2011, 5 pages.
	2.	GOLDSTEIN, T. (November 29, 1996). "The Gateway Security Model in the Java Electronic Commerce Framework," <i>The Java Electronic Commerce Framework</i> , 15 pages.
	3.	GONG, L. (May/June 1997). "Java Security: Present and Near Future," <i>IEEE Micro</i> pp. 14-19.
	4.	GONG, L. et al. (2003). "Elements of Security Policy," Chapter 5 in <i>Inside Java™ 2 Platform Security</i> , Second Edition, Addison-Wesley, Boston, MA, pp. 57-86.
	5.	GONG, L. et al. (2003). "Enforcing Security Policy," Chapter 6 in <i>Inside Java™ 2 Platform Security</i> , Second Edition, Addison-Wesley, Boston, MA, pp. 87-112.
	6.	GONG, L. et al. (December 2009). "Java Security: A Ten Year Retrospective," <i>Computer Security Applications Conference, 2009. ACSAC '09</i> , Honolulu, HI, 5 pages.
	7.	KOVED, L. et al. (1998). "The Evolution of Java Security," <i>IBM Systems Journal</i> 37(3), 17 pages.
	8.	LADUE, M.D. (1997). "When Java Was One: Threats From Hostile Byte Code," located at <http://esic.nist.gov/nisscr/1997/proceedings/104.pdf>, last visited on September 7, 2011, 12 pages.
	9.	MCGRAW, G. et al. (May 1, 1997). "Understanding the Keys to Java Security -- The Sandbox and Authentication," located at <http://www.javaworld.com/javaworld/jw-05-1997/jw-05-security.html>, last visited on August 27, 2011, 5 pages.
	10.	ORACLE AMERICA, INC. (2004). "Java™ 2 Platform Standard Ed. 5.0: Java Security Class Protection Domain," located at <http://download.oracle.com/javase/1.5.0/docs/api/java/security/ProtectionDomain.html>, last visited on August 27, 2011, 4 pages.

Examiner Signature		Date Considered	12-15-11
-----------------------	---	--------------------	----------

Approved for use through 07/31/2012 OMP 0651-0001
 U.S. Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE
 Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

37 C.F.R. 1.501 INFORMATION DISCLOSURE CITATION IN A PATENT		Complete if Known	
		Docket Number	Patent Number
		154892800500	6,192,476
		Applicant	
		LI GONG	
Issue Date		Art Unit	
February 20, 2001		3992	
Sheet	2	of	3

11.	ORACLE AMERICA, INC. (2004). "Java™ 2 Platform Standard Ed. 6: Java Security Class Protection Domain," located at <http://download.oracle.com/javase/1.5.0/docs/api/java/security/ProtectionDomain.html>, last visited on August 27, 2011, 5 pages.
12.	PAWLAN, M. (May 1998). "JDK 1.2: New Features and Functionality," located at <http://www.pawlan.com/monica/articles/jdk1.2features/>, last visited on August 3, 2011, 7 pages.
13.	SHAH, R. (December 1, 1996). "Java APIs: Playing Monopoly with Java via the JECF," <i>JavaWorld</i> , located at <http://www.javaworld.com/javaworld/jw-12-1996/jw-12-commerceapi.html>, last visited on February 13, 2010, 4 pages.
14.	WEBBASEDPROGRAMMING.COM (1998). "Web Based Programming Tutorials. Java 1.2 Unleashed," located at <http://www.webbasedprogramming.com/Java-1.2-Unleashed/ch03.htm>, last visited on August 3, 2011, 26 pages.
15.	MAZIERES, D. (August 8, 2011). "Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '447 Patent," Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 41 pages.
16.	MAZIERES, D. (August 8, 2011). "Exhibit A - Curriculum Vita. David Mazieres," Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '447 Patent, Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 8 pages.
17.	MAZIERES, D. (August 8, 2011). "Exhibit B - References Considered," Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '447 Patent, Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 7 pages.
18.	MAZIERES, D. (August 8, 2011). "Exhibit B - References Considered (Revised)," Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '447 Patent, Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 2 pages.
19.	MAZIERES, D. (August 8, 2011). "Exhibit C - U.S. Patent No. 5,958,050," Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '447 Patent, Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 10 pages.
20.	MAZIERES, D. (August 8, 2011). "Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '476 Patent," Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 49 pages.
21.	MAZIERES, D. (August 8, 2011). "Exhibit B - References Considered," Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '476 Patent, Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 7 pages.
22.	MAZIERES, D. (August 8, 2011). "Exhibit B - References Considered (Revised)," Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '476 Patent, Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 2 pages.

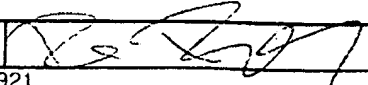
Examiner Signature		Date Considered	12-15-11
pa-1482921			

PTO/GB 42 (07-09)
 Approved for use through 07/31/2012 OMB 0651-0001
 U.S. Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE
 Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

37 C.F.R. 1.501 INFORMATION DISCLOSURE CITATION IN A PATENT		Complete if Known	
		Docket Number	Patent Number
		154892800500	6,192,476
		Applicant	
		LI GONG	
Issue Date		Art Unit	
February 20, 2001		3992	
Sheet	3	of	3

767	23.	MAZIERES, D. (August 8, 2011). "Exhibit C - U.S. Patent No. 5,412,717," Expert Report of Prof. David Mazieres, Ph.D. Regarding the Invalidity of the '476 Patent, Civil Action No. 3:10-cv-03561-WHA, presented to Oracle America, Inc. by Google, Inc., on August 8, 2011, 47 pages.
-----	-----	---

*EXAMINER: Initial if reference considered whether cited citation is in conformance with MPEP 2614. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to the third party requester. Third party requester is to place a check mark here if English language translation is attached.

Examiner Signature		Date Considered	12-15-11
--------------------	---	-----------------	----------

pa-1482921