EXHIBIT 12 - GRIFFIN IN VIEW OF CHAN

U.S. Patent No. 5,958,050
"Trusted Delegation System"
Inventors: Claire Griffin et al.
Assignee: Electric Communities
Earliest Filing Date: Sept. 24, 1996
("*Griffin*")

Patrick Chan, *The Java Class Libraries An Annotated Reference*,
Addison-Wesley (Sept. 1996) ("*Chan*")

U.S. Patent No. 6,192,476 – Claim 1	Griffin in view of Chan
1. A method for providing security, the method comprising the steps of:	*Griffin* discloses a method for providing security, i.e., "for management of trust relationships among code segments to be executed inside a trust boundary." *Griffin*, 1:23-25. "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*, 1:22–25.
detecting when a request for an action is made by a principal; and	*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 receives a request for "execution access for the class." *Griffin*, 9:61-62. "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*, 9:56–64. "The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. For example,

U.S. Patent No. 6,192,476 – Claim 1	Griffin in view of Chan
	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. Furthermore, the trust delegation system described above need not be limited to determining whether a class can be safely executed or not." *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 of the portion of code 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. 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.
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.	*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. 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, 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 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 (S1), as well as a superclass reference, a list of subclasses for the

U.S. Patent No. 6,192,476 – Claim 1	Griffin in view of Chan
	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.

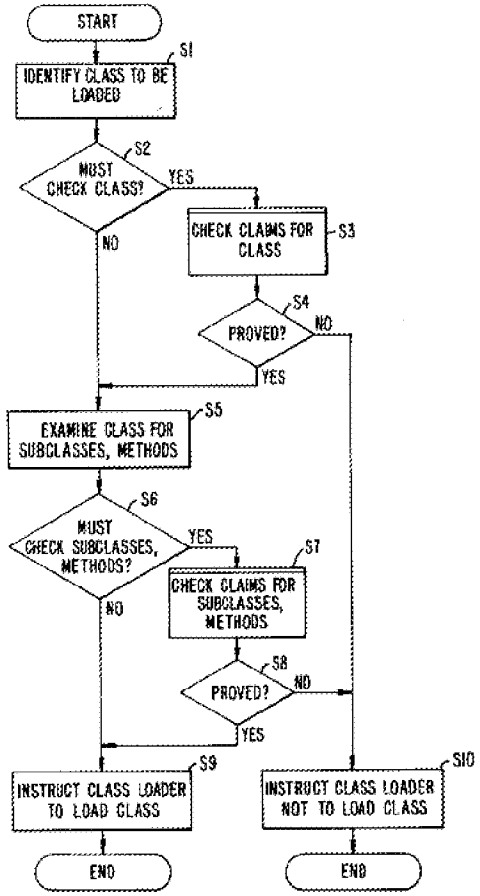


FIG. 4.

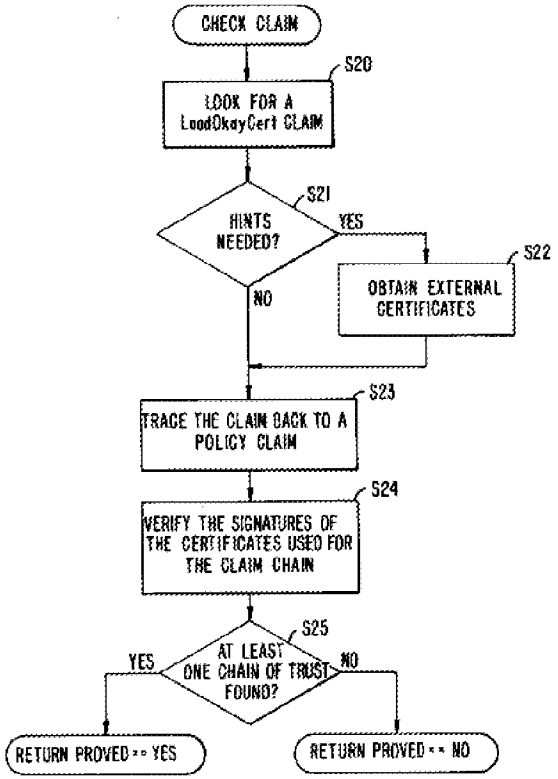


FIG. 5.

Griffin, Figs. 4–5.

"If, at step S6, the trust manager determines that a subclass or method needs to be checked,

U.S. Patent No. 6,192,476 – Claim 1	Griffin in view of Chan
	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 S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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. 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 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. 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*, either by itself or in combination with

U.S. Patent No. 6,192,476 – Claim 1	Griffin in view of Chan
	other relevant prior art, including, but not limited to *Chan*. *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. Only one security manager can be installed." *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.* 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, 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.

U.S. Patent No. 6,192,476 – Claim 2	
2. The method of claim 1, wherein:	
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; and	*Griffin* 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. *Griffin* discloses class loaders from a Java runtime system, which inherently involves a request from a thread. "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."

U.S. Patent No. 6,192,476 – Claim 2	
	Griffin, 6:48–51.
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.	*Griffin* 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. 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, 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. "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 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 S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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

U.S. Patent No. 6,192,476 – Claim 2	
	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. 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 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. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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. Only one security manager can be installed." *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.* 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()

U.S. Patent No. 6,192,476 – Claim 2	
	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. 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.

U.S. Patent No. 6,192,476 – Claim 3	
3. The method of claim 1, wherein:	
the calling hierarchy includes a first routine; and	*Chan* discloses a calling hierarchy that includes a first routine, as described in the hierarchy of calls described below with reference to the execution stack. "***Execution Stack Information*** The *execution stack* is a record of the method calls that were made from the main program to the current method. It indicates all the methods that are in progress and pending termination of the current method call. 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. The SecurityManager class provides protected methods that can be used by subclasses of the SecurityManager for this purpose." *Chan*, 1189.
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.	*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 to grant a particular access, a path of trust must be found before the access will be granted by the trust manager." *Griffin*, 9:56–63.

U.S. Patent No. 6,192,476 – Claim 3	
	"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 S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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. 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 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.

U.S. Patent No. 6,192,476 – Claim 4	
4. The method of claim 1, 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.	*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 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." *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 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 (S1), 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.

U.S. Patent No. 6,192,476 – Claim 5	
5. A method for providing security, the method comprising the steps of:	*Griffin* discloses a method for providing security, i.e., "for management of trust relationships among cod segments to be executed inside a trust boundary." "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."

U.S. Patent No. 6,192,476 – Claim 5	
	Griffin, 1:22–25.
detecting when a request for an action is made by a principal,	*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 receives a request for "execution access for the class." *Griffin*, 9:61-62. "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*, 9:56–64. "The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. For example, 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. Furthermore, the trust delegation system described above need not be limited to determining whether a class can be safely executed or not." *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 of the portion of code 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. 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.
determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy	*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. 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, execute or

U.S. Patent No. 6,192,476 – Claim 5	
associated with said principal;	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 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 (S1), 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.

U.S. Patent No. 6,192,476 – Claim 5	

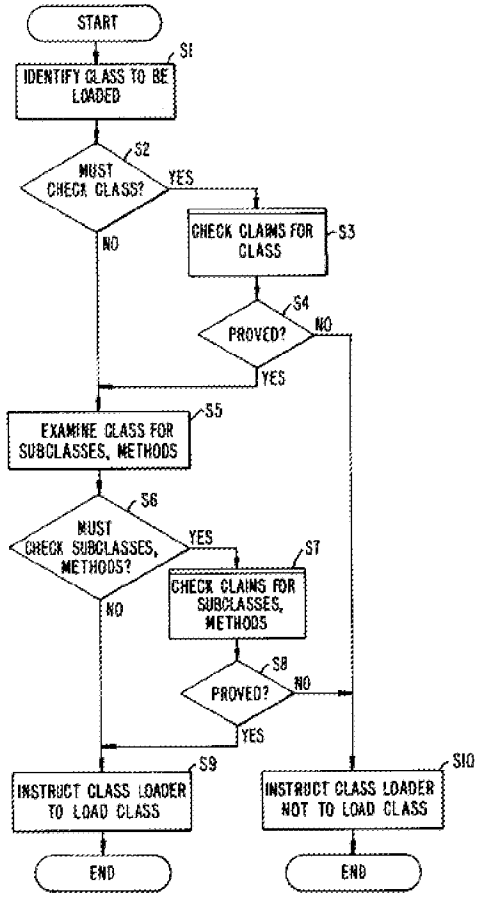


FIG. 4.

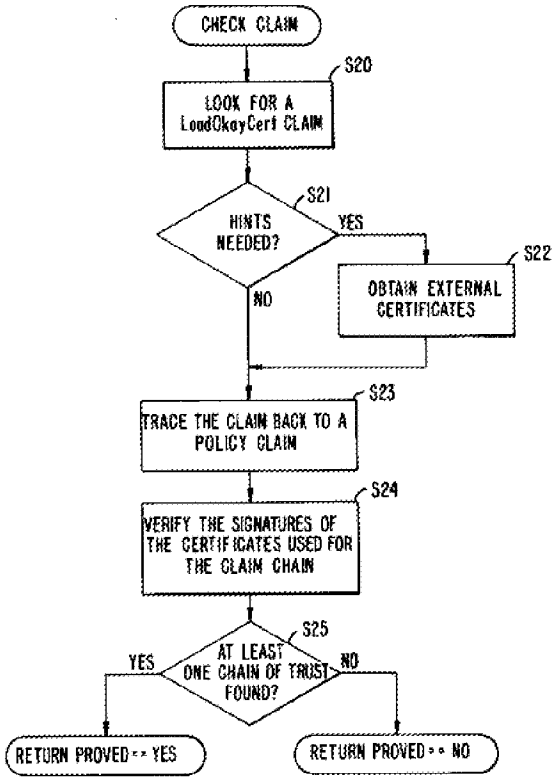


FIG. 5.

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

U.S. Patent No. 6,192,476 – Claim 5	
	loader (via an OK-to-load signal, or otherwise) to load the class (S9). If, at step S4 or step S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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. 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 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. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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

U.S. Patent No. 6,192,476 – Claim 5	
	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. Only one security manager can be installed." *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.* 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, 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.
wherein each routine of said plurality of routines is associated with a class; and	*Griffin* discloses a method wherein each routine of said plurality of routines is associated with a class. *Griffin* discloses class loaders from a Java runtime system, which inherently involves routines associated with a class. "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." *Griffin*, 6:48–51.
wherein said association between permissions and said plurality of routines is based on a second association between classes and	*Griffin* discloses a method wherein 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, a path of

U.S. Patent No. 6,192,476 – Claim 5	
protection domains.	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.

U.S. Patent No. 6,192,476 – Claim 6	
6. A method for providing security, the method comprising the steps of:	*Griffin* discloses a method for providing security, i.e., "for management of trust relationships among cod segments to be executed inside a trust boundary." "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*, 1:22–25.
detecting when a request for an action is made by a principal; and	*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 receives a request for "execution access for the class." *Griffin*, 9:61-62. "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*, 9:56–64. "The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. For example, 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. Furthermore, the trust delegation system described above need not be limited to determining whether a class can be safely executed or not." *Griffin*, 10:49–56.

U.S. Patent No. 6,192,476 – Claim 6	
	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 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.
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; and	*Griffin* discloses privilege through its reference to claims. For example, *Griffin* discloses 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, 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*, 4:49-57.
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.	*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 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. This claim element would have been obvious at the time of the invention to one of ordinary skill in the art from the teachings of *Griffin*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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

U.S. Patent No. 6,192,476 – Claim 6	
	browsers typically define a security manager and use System.setSecurityManager() to install it. Only one security manager can be installed." *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.* 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, 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.

U.S. Patent No. 6,192,476 – Claim 7	
7. The method of claim 6, 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:	*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: 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
determining whether said permission required is encompassed by at least one	

U.S. Patent No. 6,192,476 – Claim 7	
permission associated with said second routine; and	select from said plurality of routines in said calling hierarchy, or determining that said next routine is said first routine. For example, *Griffin* discloses 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 as dead ends are encountered. Eventually, the trust manager may exhaust all possible hints and have searched all available claims. *Griffin*, 9:12-16. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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. Only one security manager can be installed." *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.* 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, 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
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, 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.	

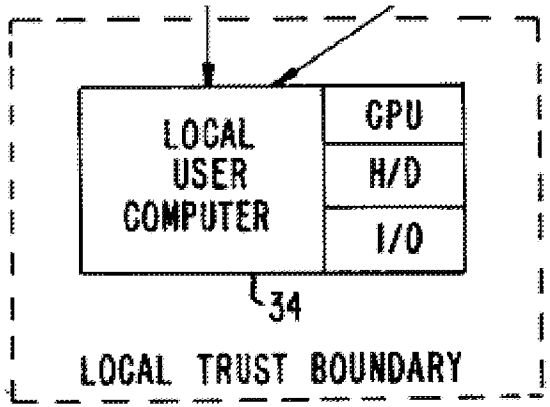
U.S. Patent No. 6,192,476 – Claim 7	
	stack, will determine whether the requested action is authorized based on the permissions associated with the cascading hierarchy of calls.

U.S. Patent No. 6,192,476 – Claim 8	
8. The method of claim 7, wherein:	*Griffin* in view of *Chan* discloses the method of claim 7, wherein: the method further includes the step of 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. 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, 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. ***Griffin*** discloses privilege through its reference to claims. For example, *Griffin* discloses 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, 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*, 4:49-57. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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
the method further includes the step of 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.	

U.S. Patent No. 6,192,476 – Claim 8	
	System.setSecurityManager() to install it. Only one security manager can be installed." *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.* 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, 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.

U.S. Patent No. 6,192,476 – Claim 9	
9. The method of claim 8, wherein the step of setting said flag associated with said first routine includes setting a flag in a frame in said calling hierarchy associated with said thread.	*Griffin* in view of *Chan* discloses the method of claim 8, wherein the step of setting said flag associated with said first routine includes setting a flag in a frame in said calling hierarchy associated with said thread. *Griffin* discloses a "flag" through its reference to claims. For example, *Griffin* discloses 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, 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*, 4:49-57. This can be applied to the hierarchy of calls as disclosed by *Chan*. For example, *Chan* discloses certain "Execution Stack Information," whereby "[t]he *execution stack* is a record

U.S. Patent No. 6,192,476 – Claim 9	
	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.*

U.S. Patent No. 6,192,476 – Claim 10	
10. A computer-readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps of:	*Griffin* discloses a computer-readable medium, e.g., a "local user computer," carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions.  *Griffin*, Fig. 2. "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*, 1:22–25.
detecting when a request for an action is made by a principal; and	*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 receives a request for "execution access for the class." *Griffin*, 9:61-62. "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

U.S. Patent No. 6,192,476 – Claim 10	
	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. "The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. For example, 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. Furthermore, the trust delegation system described above need not be limited to determining whether a class can be safely executed or not." *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 of the portion of code 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. 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.
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.	*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. 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, 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

U.S. Patent No. 6,192,476 – Claim 10	
	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 (S1), 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.

U.S. Patent No. 6,192,476 – Claim 10	

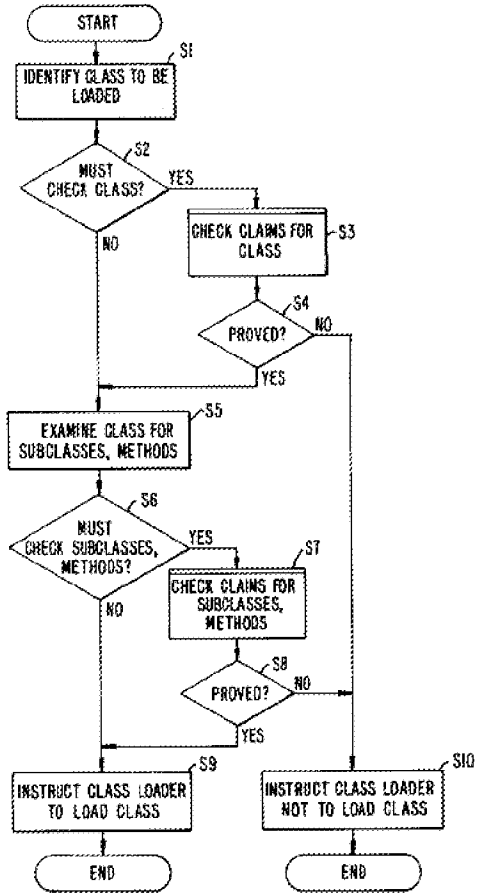


FIG. 4.

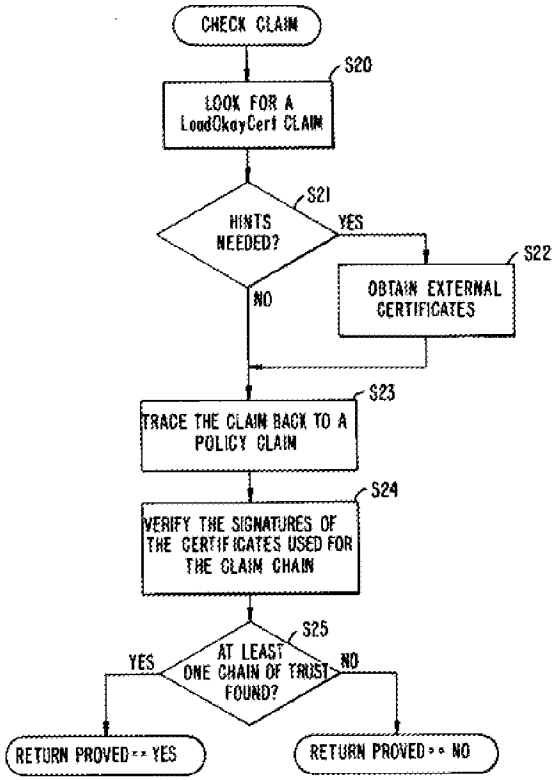


FIG. 5.

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

U.S. Patent No. 6,192,476 – Claim 10	
	loader (via an OK-to-load signal, or otherwise) to load the class (S9). If, at step S4 or step S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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. 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 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. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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

U.S. Patent No. 6,192,476 – Claim 10	
	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. Only one security manager can be installed." *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.* 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, 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.

U.S. Patent No. 6,192,476 – Claim 11	
11. The computer-readable medium of claim 10, wherein:	
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; and	*Griffin* 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. *Griffin* discloses class loaders from a Java runtime system, which inherently involves a request from a thread. "The preferred embodiment for use with Java applets includes the class loader and runtime

U.S. Patent No. 6,192,476 – Claim 11	
	executive from the Java runtime system, modified according to the present invention." *Griffin*, 6:48–51.
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.	*Griffin* 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. 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, 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. "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 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 S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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

U.S. Patent No. 6,192,476 – Claim 11	
	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. 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 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. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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. Only one security manager can be installed." *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.*

U.S. Patent No. 6,192,476 – Claim 11	
	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, 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.

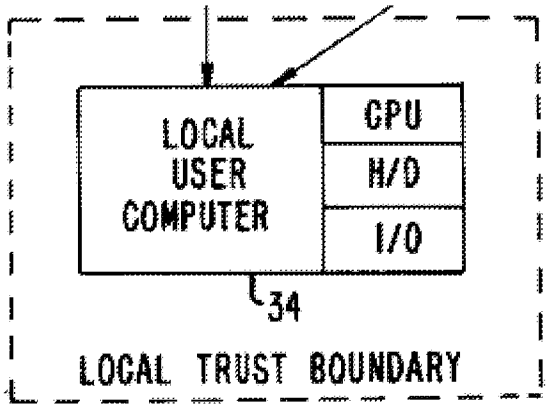
U.S. Patent No. 6,192,476 – Claim 12	
12. The computer readable medium of claim 10, wherein:	
the calling hierarchy includes a first routine; and	*Griffin* discloses a calling hierarchy that includes a first routine. See, for example, the "execution stack information" disclosure of *Chan*, referencing a calling hierarchy which necessarily involves a first routine. "***Execution Stack Information*** The *execution stack* is a record of the method calls that were made from the main program to the current method. It indicates all the methods that are in progress and pending termination of the current method call. 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. The SecurityManager class provides protected methods that can be used by subclasses of the SecurityManager for this purpose." *Chan*, 1189.
the step of determining whether said action is authorized further includes determining whether a permission	*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. For example, *Griffin* discloses that "[i]f it

U.S. Patent No. 6,192,476 – Claim 12	
required to perform said action is encompassed by at least one permission associated with said first routine.	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*, 9:56–63. "If, at step S6, the trust manager determines that a subclass or method needs to be checked, that is done (S7) by the process 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 S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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. 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 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

U.S. Patent No. 6,192,476 – Claim 12	
	portion of code is allowed by the policy rules given the potential resource use, the code supplier and applicable certificates." *Griffin*, 3:33–57.

U.S. Patent No. 6,192,476 – Claim 13	
13. The computer readable medium of claim 10, 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.	*Griffin* 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. 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, 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 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 (S1), 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.

U.S. Patent No. 6,192,476 – Claim 14	
14. A computer-readable medium bearing instructions for providing	*Griffin* discloses a computer-readable medium, e.g., a "local user computer," bearing instructions for providing security.

U.S. Patent No. 6,192,476 – Claim 14	
security, the instructions including instructions for performing the steps of:	 *Griffin*, Fig. 2. "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*, 1:22–25.
detecting when a request for an action is made by a principal;	*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 receives a request for "execution access for the class." *Griffin*, 9:61-62. "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*, 9:56–64. "The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. For example, 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

U.S. Patent No. 6,192,476 – Claim 14	
	decision is used to decide whether or not to execute the code. Furthermore, the trust delegation system described above need not be limited to determining whether a class can be safely executed or not." *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 of the portion of code 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. 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.
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;	*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. 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, 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. "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*, 9:56–64. "The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. For example, 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. Furthermore, the trust delegation system described above need not be limited to determining whether a class can be

U.S. Patent No. 6,192,476 – Claim 14	
	safely executed or not." *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 of the portion of code 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. 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. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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. Only one security manager can be installed." *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.* 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, they may need to inspect the execution stack to find out information about the

U.S. Patent No. 6,192,476 – Claim 14	
	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.
wherein each routine of said plurality of routines is associated with a class; and	*Griffin* discloses each routine of said plurality of routines is associated with a class. *Griffin* discloses class loaders from a Java runtime system, which inherently involves routines associated with a class. "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." *Griffin*, 6:48–51.
wherein said association between permissions and said plurality of routines is based on a second association between classes and protection domains.	*Griffin* discloses that said association between permissions and said plurality of routines is based on a second association between classes and protection domains. For example, *Griffin* discloses a unique identifier for a class as well as a superclass reference, as described in further detail below: "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 (S1), 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.

U.S. Patent No. 6,192,476 – Claim 14	

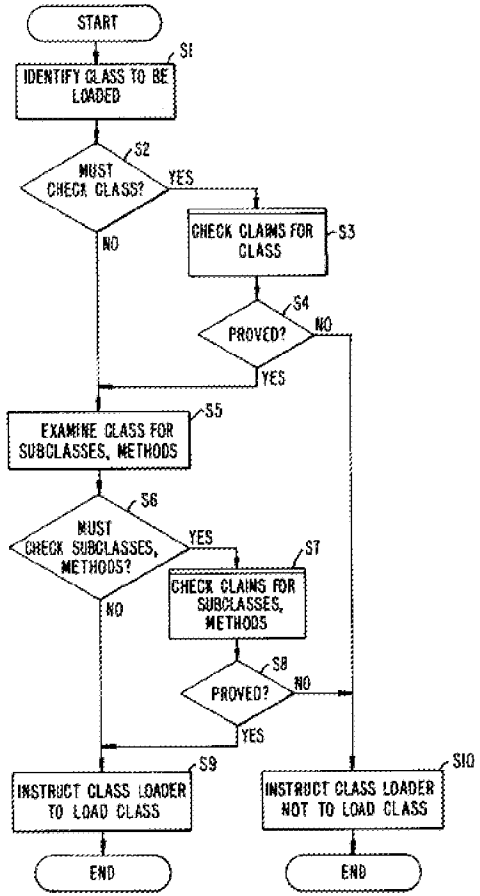


FIG. 4.

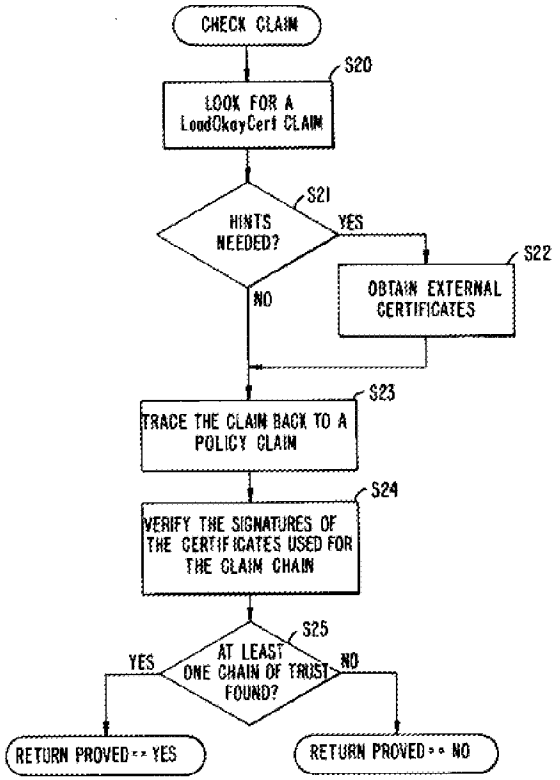


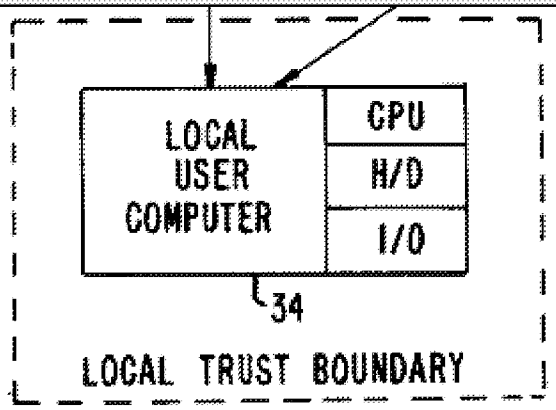
FIG. 5.

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

U.S. Patent No. 6,192,476 – Claim 14	
	loader (via an OK-to-load signal, or otherwise) to load the class (S9). If, at step S4 or step S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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. 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 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.

U.S. Patent No. 6,192,476 – Claim 15	
15. A computer-readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more	*Griffin* discloses a computer-readable medium, e.g., a "local user computer," carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions.

U.S. Patent No. 6,192,476 – Claim 15	
instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps of:	 *Griffin*, Fig. 2. "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*, 1:22–25.
detecting when a request for an action is made by a principal; and	*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 receives a request for "execution access for the class." *Griffin*, 9:61-62. "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*, 9:56–64. "The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. For example, 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. Furthermore, the trust

U.S. Patent No. 6,192,476 – Claim 15	
	delegation system described above need not be limited to determining whether a class can be safely executed or not." *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 of the portion of code 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. 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.
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; and	*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. *Griffin* discloses privilege through its reference to claims. For example, *Griffin* discloses 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, 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*, 4:49-57. "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 (S1), 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.

U.S. Patent No. 6,192,476 – Claim 15	

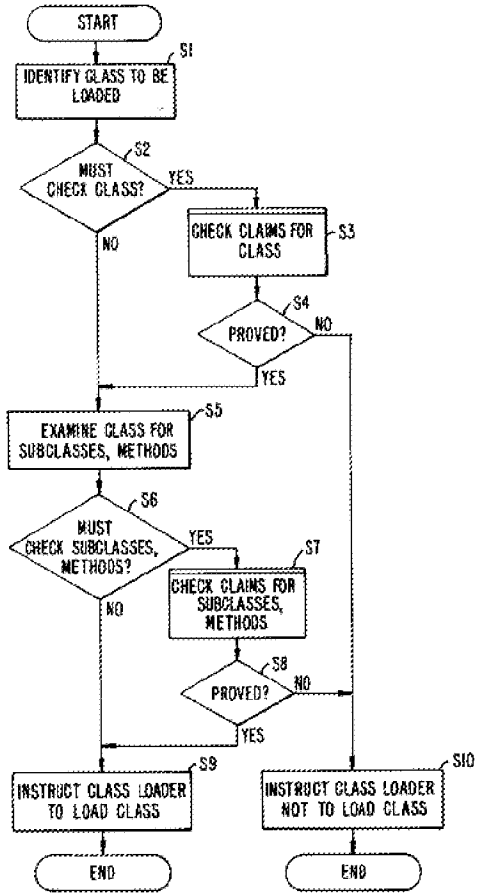


FIG. 4.

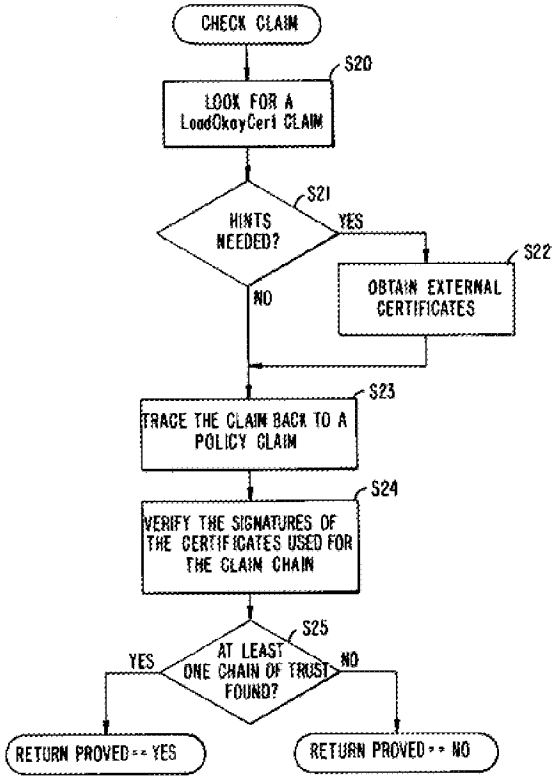


FIG. 5.

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

U.S. Patent No. 6,192,476 – Claim 15	
	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 S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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. 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 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. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *Chan* discloses that "[a]

U.S. Patent No. 6,192,476 – Claim 15	
	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. Only one security manager can be installed." *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.* 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, 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.
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	*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 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. *Griffin* discloses privilege through its reference to claims. For example, *Griffin* discloses that "[t]he process of 'proving' a claim is one of verifying the claimant, a level of trust in the

U.S. Patent No. 6,192,476 – Claim 15	
second routine is invoked after said first routine, wherein said second routine is a routine for performing said requested action.	claimant and the authorization of that claimant, 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*, 4:49-57. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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. Only one security manager can be installed." *Chan*, 1188.

U.S. Patent No. 6,192,476 – Claim 16	

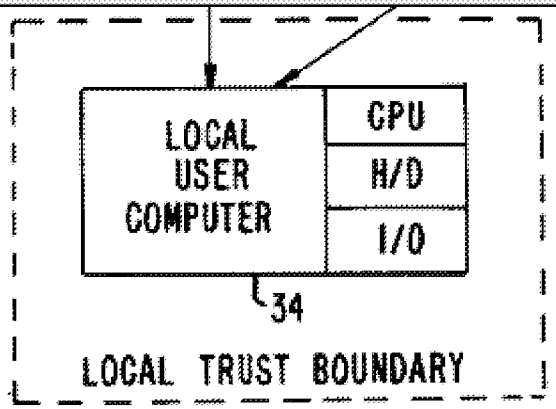
U.S. Patent No. 6,192,476 – Claim 16	
16. The computer readable medium of claim 15, wherein the step of determining whether said permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first 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; 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: 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* in view of *Chan* discloses the computer readable medium of claim 15, 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; 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: 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. For example, *Griffin* discloses privilege through its reference to claims. For example, *Griffin* discloses 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, 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*, 4:49-57. *Griffin* also discloses that "[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, 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. This iterative process will be compiled until exhaustion: The trust manager will keep searching for a path through the web of claims, using hints as needed, moving up and down along a path as dead ends are encountered. Eventually, the trust manager may exhaust all possible hints and have searched all available claims. *Griffin*, 9:12-16.

U.S. Patent No. 6,192,476 – Claim 17	
17. The computer readable medium of claim 16, wherein: the computer readable medium further comprises one or more instructions for performing the step of 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.	*Griffin* discloses the computer readable medium of claim 16, wherein: the computer readable medium further comprises one or more instructions for performing the step of 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. *Griffin* discloses privilege through its reference to claims. For example, *Griffin* discloses 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, 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*, 4:49-57. The trust manager will keep searching for a path through the web of claims, using hints as needed, moving up and down along a path as dead ends are encountered. Eventually, the trust manager may exhaust all possible hints and have searched all available claims. *Griffin*, 9:12-16.

U.S. Patent No. 6,192,476 – Claim 18	
18. The computer readable medium of claim 17, wherein the step of setting said flag associated with said first routine includes setting a flag in a frame in said calling hierarchy associated with said thread.	*Griffin* in view of *Chan* discloses the computer readable medium of claim 17, wherein the step of setting said flag associated with said first routine includes setting a flag in a frame in said calling hierarchy associated with said thread. *Griffin* discloses a "flag" through its reference to claims. For example, *Griffin* discloses 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, 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*, 4:49-57. And *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,

U.S. Patent No. 6,192,476 – Claim 18	
	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. Only one security manager can be installed." *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.* 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, 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.

U.S. Patent No. 6,192,476 – Claim 19	
19. A computer system comprising:	
a processor; a memory coupled to said processor;	*Griffin* discloses a processor and a memory coupled to said processor by way of reference to a "local user computer."

U.S. Patent No. 6,192,476 – Claim 19	
	 *Griffin*, Fig. 2.
said processor being configured to detect when a request for an action is made by a principal; and	*Griffin* discloses said processor being configured to detect when a request for an action is made by a principal. *Griffin* does this by, e.g., checking the relevant security provisions when it receives a request for "execution access for the class." *Griffin*, 9:61-62. "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*, 9:56–64. "The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. For example, 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. Furthermore, the trust delegation system described above need not be limited to determining whether a class can be safely executed or not." *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

U.S. Patent No. 6,192,476 – Claim 19	
	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 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.
said processor being configured to respond to detecting the request by 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.	*Griffin* discloses said processor being configured to respond to detecting the request by 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. 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, 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 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 (S1), 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.

U.S. Patent No. 6,192,476 – Claim 19	

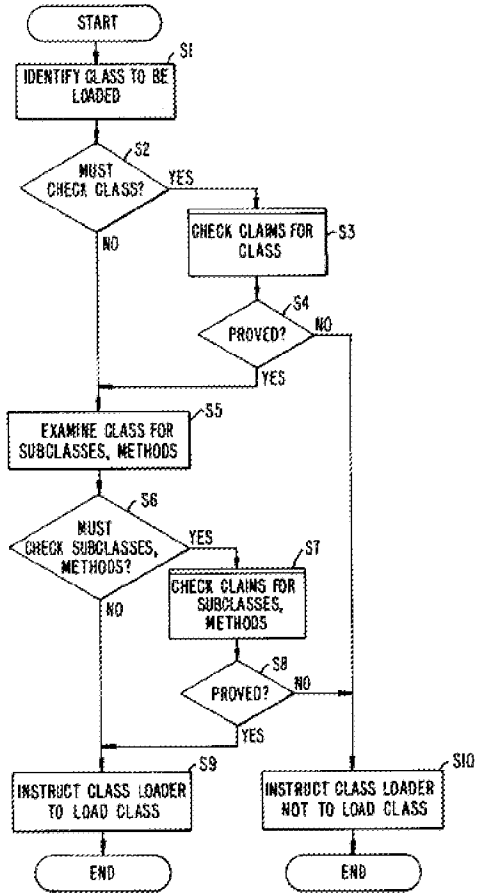


FIG. 4.

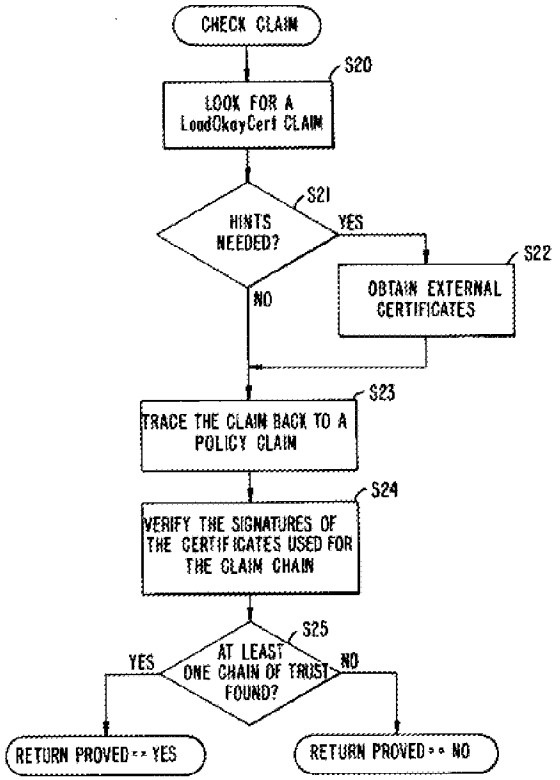


FIG. 5.

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

U.S. Patent No. 6,192,476 – Claim 19	
	loader (via an OK-to-load signal, or otherwise) to load the class (S9). If, at step S4 or step S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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. 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 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. 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*, either by itself or in combination with other relevant prior art, including, but not limited to *Chan*. *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

U.S. Patent No. 6,192,476 – Claim 19	
	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. Only one security manager can be installed." *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.* 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, 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.

U.S. Patent No. 6,192,476 – Claim 20	
20. The computer system of claim 19, wherein:	
the calling hierarchy includes a first routine; and	*Chan* discloses a calling hierarchy that includes a first routine, as disclosed in the "execution stack" with multiple routines, discussed below: "***Execution Stack Information*** The *execution stack* is a record of the method calls that were made from the main program to

U.S. Patent No. 6,192,476 – Claim 20	
	the current method. It indicates all the methods that are in progress and pending termination of the current method call. 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. The SecurityManager class provides protected methods that can be used by subclasses of the SecurityManager for this purpose." *Chan*, 1189.
said processor is configured to determine whether said action is authorized by determining whether a permission required to perform said action is encompassed by at least one permission associated with said first routine.	*Griffin* discloses a processor to determine whether said action is authorized by determining whether a permission required to perform said action is encompassed by at least one permission associated with said first routine. For example, "[i]f 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*, 9:56–63. "If, at step S6, the trust manager determines that a subclass or method needs to be checked, that is done (S7) by the process 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 S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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. "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

U.S. Patent No. 6,192,476 – Claim 20	
	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. 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 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.

U.S. Patent No. 6,192,476 – Claim 21	
21. The computer system of claim 19, wherein	
said processor is configured to determine whether said action is authorized by determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy.	*Griffin* discloses a processor configured to determine whether said action is authorized by 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]f 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*, 9:56–63. "If, at step S6, the trust manager determines that a subclass or method needs to be checked, that is done (S7) by the process 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 S8 the trust manager cannot prove the required claim, the trust manager instructs the class loader not to load the class (S10). 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

U.S. Patent No. 6,192,476 – Claim 21	
	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. "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. 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 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.