Identity-based Access Control

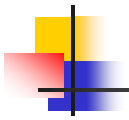
- The kind of access control familiar from operating systems like Unix or Windows based on user identities
- This model originated in 'closed' organisations ('enterprises') like universities, research labs, that have authority over its members.
- Members (users) can be physically located.
- Access control policies refer naturally to user identities.
- Audit logs point to users who can be held accountable.
- Access control seems to require by definition that identities of persons are verified.

Other Aspects

- Access rules are local: no need to search for the rule that should be applied since stored in ACL with the object.
- Enforcement of rules is centralized: reference monitor is independent when making a decision.
- Simple access operations: read, write, execute; single subject per rule; no rules based on object content.
- Homogeneity: same organisation defines organizational security policy and automated security policy.

Changes in the 1990s

- Internet connections to parties never met before:
  - 'identity' cannot be part of our access rules.
  - not always able to hold them accountable.
- Java sandbox shows it is not necessary to refer to users when describing or enforcing access control.
- Access controlled at level of applets, not at granularity of read/write/execute.
- Instead of asking who made the request, ask what to do with it.



What changed with the web?

- Separation of program and data is blurred; executable content (applets, scripts) embedded in interactive web pages that can process user input.
- Computation moved to the client who needs protection from rogue content providers.
  - Lesson from early PC age: floppy disks from arbitrary sources were the route for computer virus infections.
- Client asked to make decisions on security policy and on enforcing security: end user becomes system administrator and policy maker.
- Browser becomes part of the TCB.

Changes in the Environment

- When organisations collaborate, access control can be based on more than one policy and potential conflicts between policies have to be addressed.
- How to export security identifiers from one system into another system?
- Decision on access requests may be made by entity other than the one enforcing it
- How does a user know which credentials to present?

Code-based Access Control

- If not possible to rely on principal who requests an access control decision, look at the request itself.
- Requests can be programs, rather than elementary read/write instructions.
- Code-based access control: access control where permissions are assigned to (parts of) code.
- Major examples: Java security model, .NET security framework (code access control).

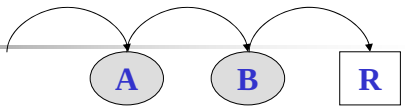
Access Control Parameters

- Security attributes of code could be:
  - Site of code origin: local or remote?
  - URL of code origin: intranet or Internet?
  - Code signature: signed by "trusted" author?
  - Code proof: code author provides proof of security properties
  - Identity of sender: principal the code comes from
  - …

Questions central to code-based access control.

- In code-based access control, when process calls another function, access decisions refer to access rights assigned to that function.
- Should calling process also **delegate** some of its access rights to process executing the function being called?
- Should calling process **limit** access rights of the function executing the program being called?
- Which privileges should be valid when one function calls another function?

Call Chains



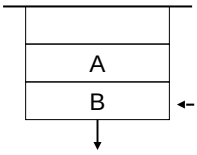
- Example 1: function A has access right to resource R, B does not; A calls B, B requests access to R: Should access be granted?
  - Conservative answer is 'no', but A could explicitly delegate the access right to B.
- Example 2: function B has access right to resource R, A does not; A calls B, B requests access to R: Should access be granted?
  - Conservative answer is 'no', but B could explicitly assert its access right.

Enforcing Policies

- How to compute current permissions granted to code?
- Access decisions should know about entire call chain.
- Information about callers maintained on call stack used by Java VM for managing executions.
- Design decision: re-use call stack for policy evaluation.
- Lazy evaluation: evaluate granted permissions just when a permission is required to access a resource.

Dynamic Stack Inspection

- Record, for each stack frame, the security permissions of the function.
- Rights of final caller are computed as the intersection of permissions for all entries on call stack.



effective rights =
rights(B) ∩ rights(A) ∩ …

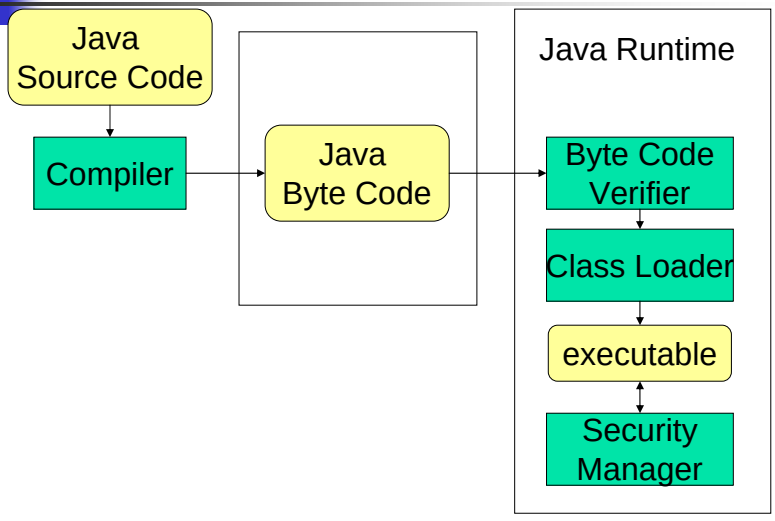
Limits of Stack Inspection

- Access control explained in terms of runtime stack for implementation reasons (lazy evaluation).
  - Performance? Common optimizations are disabled.
  - Security: What guaranteed by stack inspection? Hard to relate to high-level security policies.
- Two concerns for developers:
  - Untrusted component may take advantage of my code.
  - Permissions may be missing when running my code.
- Stack inspection blind to many control and data flows:
  - Parameters, results, mutable data, objects, inheritance, callbacks, events, exceptions, concurrency…
- Each case requires a specific discipline or mechanism.

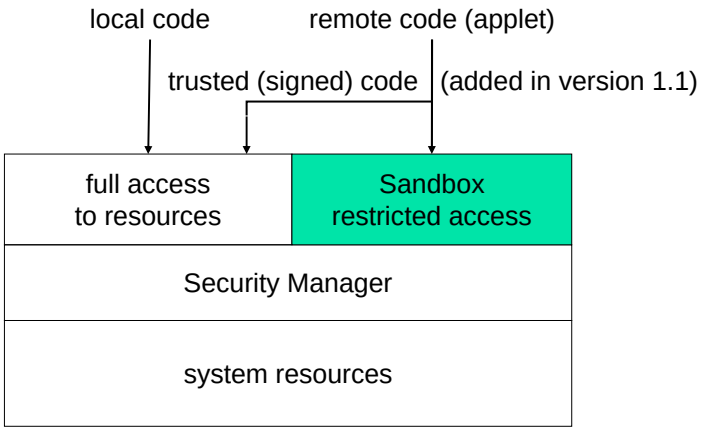
Java Security

- Java: strongly typed - type safe - object-oriented general purpose programming language.
- Static (and dynamic) type checking to examine whether arguments received during execution always of correct type.
- Security: no pointers arithmetic; memory access through pointers one of main causes for security flaws in C or C++.
- Java source code translated into machine independent byte code (similar to assembly language) and stored in class files.
- *Platform specific* virtual machine interprets byte code translating it into machine specific instructions.
- When running a program, a Class Loader loads any additional classes required.
- Security Manager enforces the given security policy.

Java Execution Model

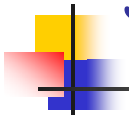


JDK 1.1 Security Model



Discussion

- Basic policy inflexible:
  - Local/signed code is unrestricted.
  - Applet/unsigned code is restricted to sandbox.
  - No intermediate level: how to give some privileges to a home banking application?
- Local/remote is not a precise security indicator:
  - Parts of local file system could reside on other machines
  - Downloaded software becomes "trusted" once cached or installed on local system.
- For more flexible security policies a customized security manager needed to be implemented.
  - Requires security AND programming skills.



Java 2 Security Model

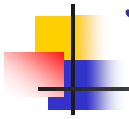
- Java 2 security model no longer based on distinction between local code and applets, and applets and applications controlled by same mechanisms.
- Reference monitor of Java security model performs fine-grained access control based on security policies and permissions.
- Policy definition separated from policy enforcement.
- Single method checkPermissions() handles all security checks.

Byte Code Verifier

- Analyzes Java class files: performs syntactic checks, uses theorem-provers and data flow analysis for static type checking.
- There is still dynamic type checking at run time
- Verification guarantees properties like:
  - Class file is in proper format.
  - Stacks will not overflow.
  - All operands have arguments of correct type.
  - No data conversion between types.
  - All references to other classes are legal.

Class Loaders

- Protect integrity of run time environment; applets not allowed to create their own Class Loaders and to interfere with each other.
- Vulnerabilities in a class loader are particularly security critical (if exploited by attacker to insert rogue code).
- Each Class Loader has own name space; each class labeled with Class Loader that installed it.

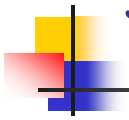


Security Policies

- **Security policy:** maps set of properties characterizing running code to set of access permissions granted to the code.
- Code characterized by CodeSource:
  - origin (URL)
  - digital certificates
- Permissions contain target name and set of actions and granted to protection domains:
  - Classes and objects belong to protection domains and 'inherit' the granted permissions.
  - Each class belongs to one and only one domain.

Security Manager

- **Security Manager**: reference monitor in JVM; security checks defined in AccessController class.
  - Uniform access decision algorithm for all permissions.
- Access (normally) only granted if all methods in the current sequence of invocations have the required permissions ('stack walk').
- Controlled invocation: privileged operations; doPrivileged() tells the Java runtime to ignore the status of the caller.



Summary

- Java 2 security model flexible and feature-rich; it gives a framework but does not prescribe a fixed security policy.
- JAAS (Java Authentication and Authorization Service) adds user-centric access control.
- Sandbox enforces security at service layer; security can be undermined by access to layer below:
  - users running applications other than web browser.
  - attacks by breaking the type system.