New
Access Control Paradigms

Introduction

Internet and the World Wide Web have brought large many ‘security unaware’ users into direct contact with new IT applications.

Mobile code from the Internet is running on client machines.

Electronic commerce promises new business opportunities.

We are facing considerable change in the way IT systems are being used; are the old security paradigms still fit or do we need new policies and new enforcement mechanisms?

Objectives

Explore new paradigms for access control.

Explain background and rationale for the move to code-based access control.

Present stack walking as the main security enforcement algorithm used in code-based access control.

Give an introduction to the Java security model and the .NET security framework.

Agenda

Access Control – Origins

Code-based access control

Java and .NET security models

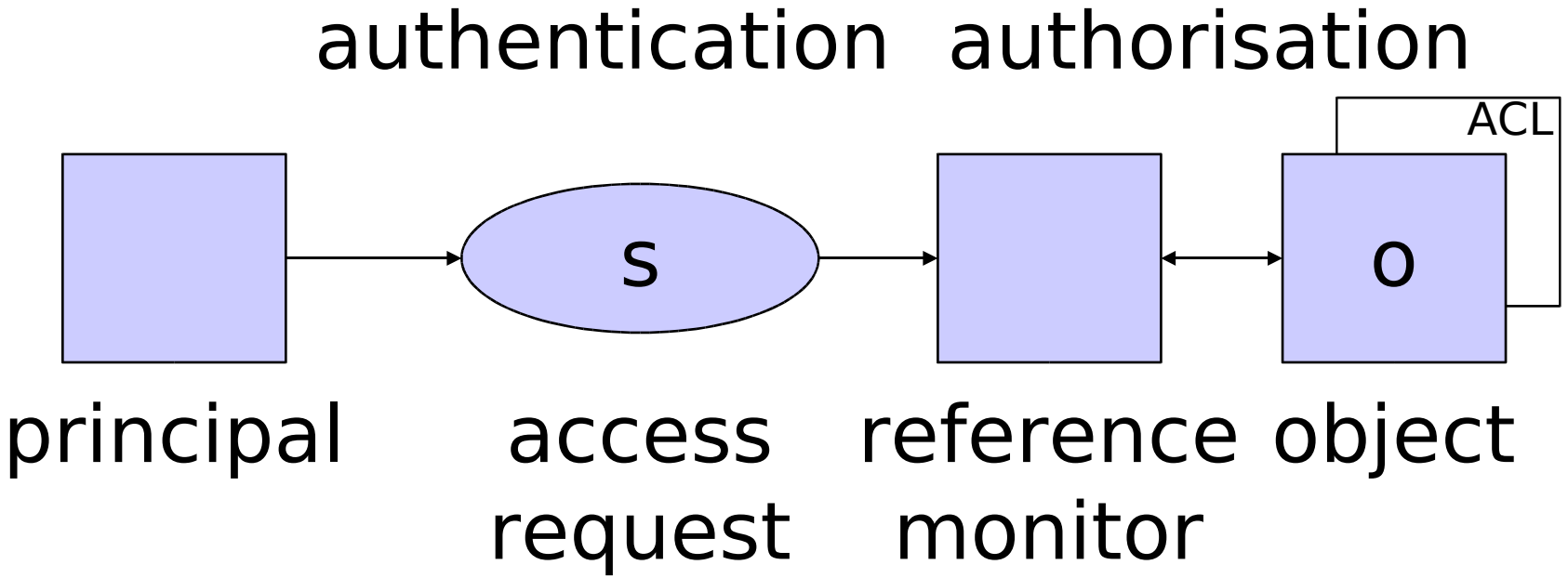
Cookies

SPKI: PKI & access control

Trust Management Systems

Digital Rights Management

Access Control – Origins



B. Lampson, M. Abadi, M. Burrows, E. Wobber: Authentication in Distributed Systems: Theory and Practice, ACM Transactions on Computer Systems, 10(4), pages 265-310, 1992

Identity-based Access Control

Access control based on user identities.

The kind of access control familiar from operating systems like Unix or Windows.

Do not confuse the ‘identity’ of a person with a user identity (uid) in an operating systems; a uid is just a unique identifier that need not correspond to a real person (e.g. ‘root’).

RBAC = IBAC + one level of indirection.

Fact File

This model originated in ‘closed’ organisations (‘enterprises’) like universities, research labs.

The organisation has authority over its members.

The members (users) can be physically located.

Access control policies refer naturally to user identities: ACEs associated with known people.

Audit logs point to users who can be held accountable.

Access control seems to require by definition that identities of persons are verified.

Biometrics: strong identity-based access control?

Further Aspects

Access rules are local: no need to search for the rule that should be applied; the rule is stored as an ACL with the object.

Enforcement of rules is centralized: reference monitor does not consult other parties when making a decision.

Simple access operations: read, write, execute; single subject per rule; no rules based on object content.

Homogeneity: the same organisation defines organizational and automated security policy.

Changes in the Last Decade

The Internet connects us to parties we never met before:

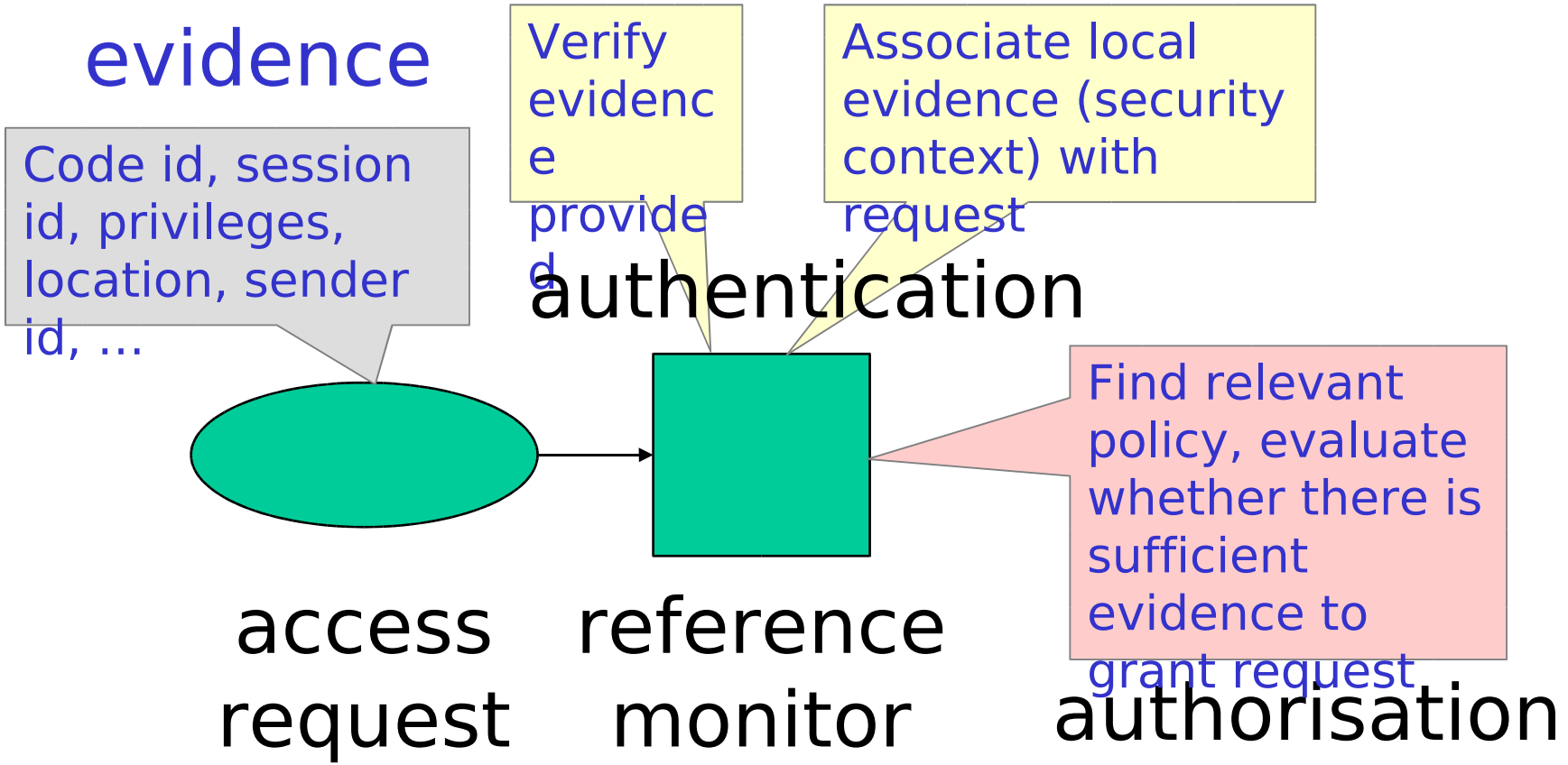
- Their ‘identity’ can hardly figure in our access rules.
- We are not always able to hold them accountable.

Java sandbox: it is not necessary to refer to users when describing or enforcing access control.

Access controlled at the level of applets, not at the granularity of read/write/execute.

Instead of asking who made the request, ask what to do with it.

Access Control in an ‘Open’ World



Changes: The Web

Executable content (applets) blurs separation between program and data.

Computation moved to the client who needs protection from content providers.

Lesson of the early PC age: floppy disks from arbitrary sources were the route for computer virus infections.

As computation moves to the client, the client is asked to make decisions on security policy, and on enforcing security.

The browser becomes part of the TCB.

Changes in the Environment

When organisations collaborate, access control can be based on more than one policy.

Potential conflicts between policies have to be addressed.

How to export security identifiers from one system into another system?

Decisions on access requests may be made by an entity other than the one enforcing the decision.

How does a user know which credentials to present?

Splitting the Reference Monitor

Policy administration point (PAP): creates a policy or policy set.

Policy decision point (PDP): evaluates applicable policy and renders an authorization decision.

Policy enforcement point (PEP): performs access control, by making decision requests and enforcing authorization decisions.

Policy information point (PIP): acts as a source of attribute values.

Changes in Mechanisms

Locally stored access rules can be placed in protected memory segments.

Access rules sent to remote sites need cryptographic protection.

Locally stored access rights of principals can be placed in protected memory segments.

Access rights of principals sent to remote sites need cryptographic protection.

Blurred difference between rules and rights.

Code-based Access Control

If we cannot rely on the principal who makes the request for access control decisions, we can only 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 code.

Major examples: Java security model, .NET security framework

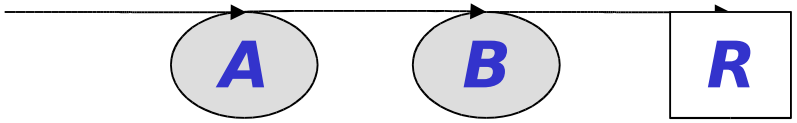
Check that the caller's allocated (granted) permissions match the required permissions.

Access Control Parameters

Security attributes associated with code can be:

- site of code origin: local or remote?
- URL of code origin: intranet or Internet?
- code signature: signed by trusted author?
- code identity: approved ('trusted') code?
- code proof: code author provides proof of security properties;
- identity of sender: principal the code comes from;
- …

Call Chains

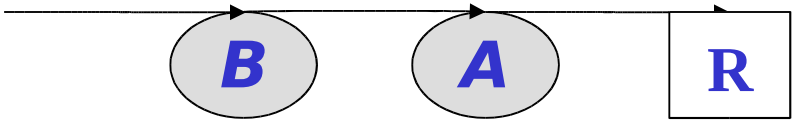


Which privileges should be valid when one method calls another method?

Example: Method *A* has access right to resource *R*, *B* does not; *A* calls *B*, *B* requests access to *R*: should access be granted?

The conservative answer is ‘no’, but *A* could explicitly delegate the access right to *B*.

Call Chains



Example: Method *A* has access right to resource *R*, *B* does not; *B* calls *A*, *A* requests access to *R*: should access be granted?

Confused deputy problem: an ‘untrusted’ entity asks a ‘trusted’ entity to do something illegal.

The conservative answer is ‘no’, but *A* could explicitly assert its access right.

Enforcing Policies

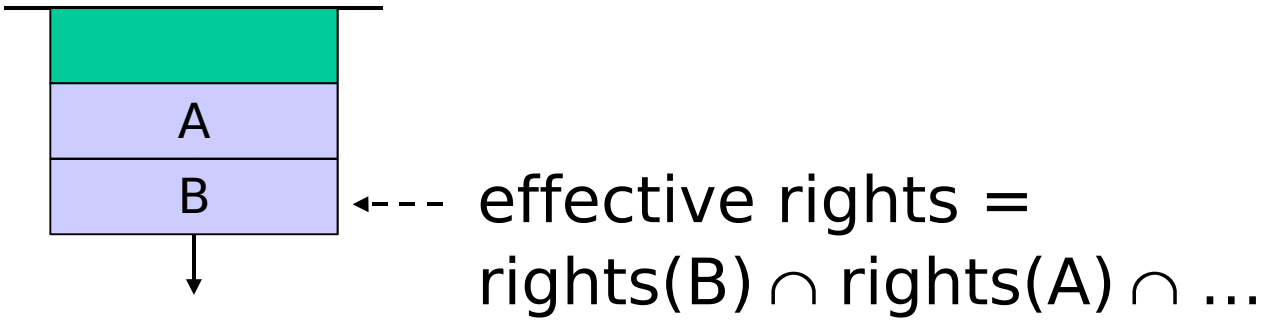
To compute the current permissions granted to code we have to know the entire call chain when making access decisions.

Java VM and .NET CLR use a call stack to manage executions; information about calling methods can be found there.

Lazy evaluation: only evaluate granted permissions when a permission is actually required to access a resource.

Stack Walk

The rights of the final caller are computed as the intersection of the rights for all entries on the call stack.



Limits of Stack Inspection

Access control makes use of the runtime stack:

- Performance? Common optimizations are disabled.
- Security: What is guaranteed by stack inspection?
  Hard to relate to high-level security policies.

Two concerns for programmers:

- Untrusted component may take advantage of my code.
- Permissions may be missing when running my code.

Stack inspection is 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.

History-Based Access Control

Don’t be lazy, keep track of callers’ rights proactively (eager evaluation).

Static rights (S) associated with each piece of code at load time.

Current rights (D) associated with each execution unit, updated automatically at execute time
(D := D ∩ S).

Controlled modifications of current rights using “grant” and “accept” programming patterns.

Java Security

Java Security

Java: strongly typed object-oriented language; general purpose programming language.

Java is type safe; the type of a Java object is indicated by the class tag stored with the object

Static (and dynamic) type checking to examine whether the arguments an operand may get during execution are always of the correct type.

Security advantage: no pointers arithmetic; memory access through pointers is one of the main causes for security flaws in C or C++.

Java – Overview

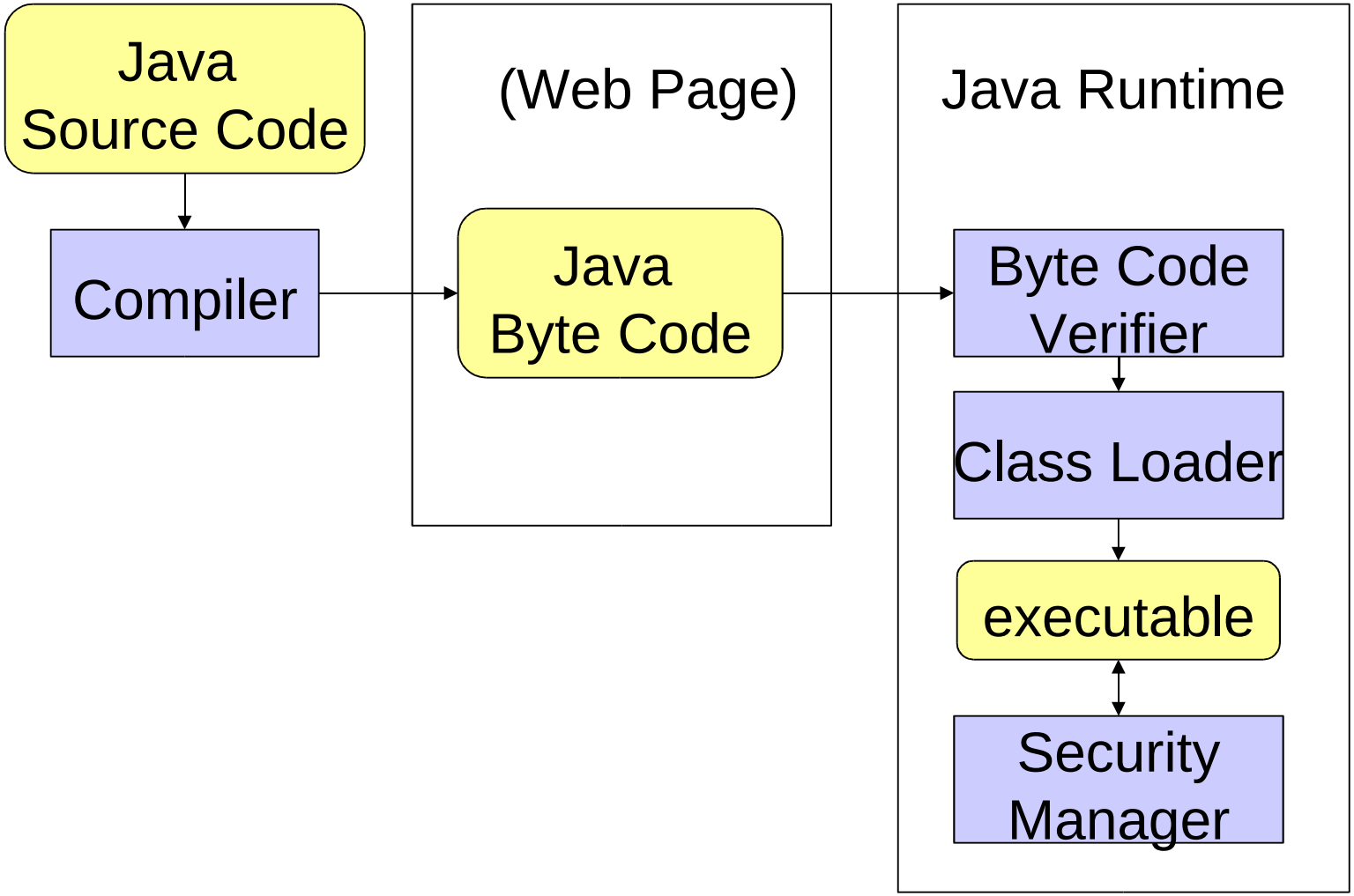
Java source code is translated into machine independent byte code (similar to an assembly language) and stored in class files.

A platform specific virtual machine interprets the byte code translating it into machine specific instructions.

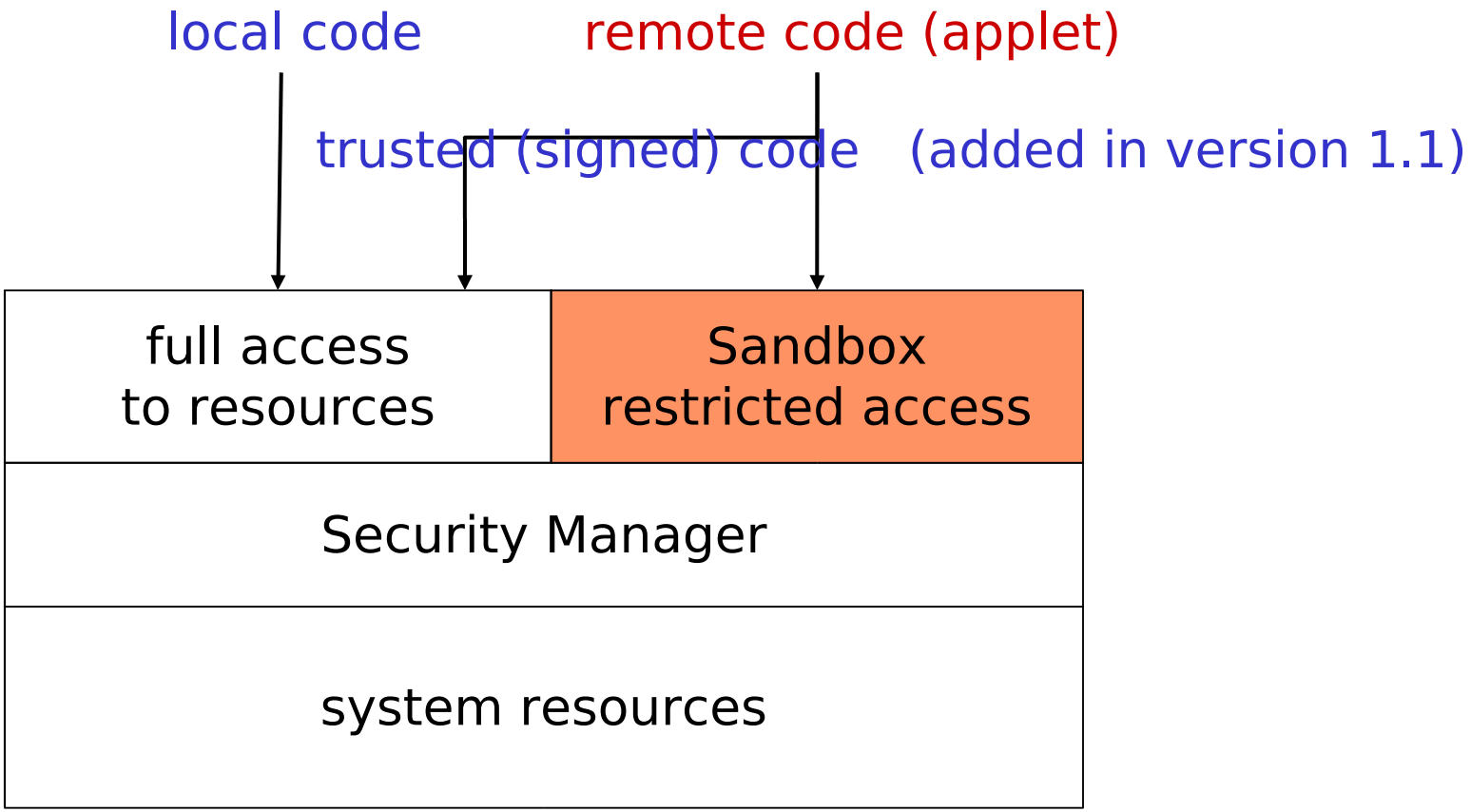
When running a program, a Class Loader loads any additional classes required.

The Security Manager enforces the given security policy.

The Java Execution Model



JDK 1.1 Security Model



Discussion

Basic policy is quite 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 the local file system could reside on other machines;
- Downloaded software becomes “trusted” once it is cached or installed on the 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 the distinction between local code and applets.

Applets and applications controlled by the same mechanisms.

The reference monitor of the Java security model performs fine-grained access control based on security policies and permissions.

Policy definition separated from policy enforcement.

A single method checkPermissions() handles all security checks.

Byte Code Verifier

Analyses 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:

- The class file is in the proper format.
- Stacks will not overflow.
- All operands have arguments of the correct type.
- There will be no data conversion between types.
- All references to other classes are legal.

Class Loaders

Protect integrity of the run time environment; applets are not allowed to create their own Class Loaders and should not interfere with each other.

Vulnerabilities in a class loader are particularly security critical as they may be exploited by an attacker to insert rogue code.

Each Class Loader has its own name space; each class is labeled with the Class Loader that has installed it.

Security Policies

Security policy: maps a set of properties that characterizes running code to a set of access permissions granted to the code.

Code characterized by CodeSource:

- origin (URL)
- digital certificates

Permissions contain target name and a set of actions.

Level of indirection: permissions 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 the 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

The Java 2 security model is 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 the service layer; security can be undermined by access to the layer below:

- users running applications other than the web browser.
- attacks by breaking the type system.

.NET Security Framework

.NET Components

Common Language Runtime (CLR): common runtime system for a variety of programming languages; loads and executes code, performs security checks (similar to JVM).

C#: Type-safe programming language developed by Microsoft (similarities to Java; builds to some extent on experiences gained from using Java.)

MSIL: Microsoft Intermediate Language (conceptually similar to Java byte code.)

Managed Code

Native code: Code compiled to machine language for a specific hardware platform; not controlled by the CLR.

Unmanaged code = native code

Managed code: Code compiled to run in the .NET framework; controlled by the CLR.

Assembly: logical unit of IL code in the .NET framework, usually a single managed DDL or EXE file.

Access Control Model

Evidence: information about the origin of code.

Authenticate code identity: collect and verify evidence about a piece of code (an assembly).

Authorize code, not users to access resources; security policies refer to evidence (about assemblies).

Enforce authorisation decisions made on individual pieces of code, such as assemblies.

Default Evidence Classes

Application Directory

Hash

Permission Request Evidence: states the permissions an assembly must have to run.

Publisher

Web Site

Strong Name

URL

Zone: security zone as in Internet Explorer

Strong Names

Assemblies are referenced through names.

Strong names: include identity of the publisher (but no third party certificate!).

- Creates separate name spaces for assembly names.

Assemblies protected by digital signatures:

- Publisher's public key given in the metadata.
- Digital signature computed and written into assembly during compilation.
- Provides origin authentication & data integrity.

Associating Evidence

Evidence applies to executing code.

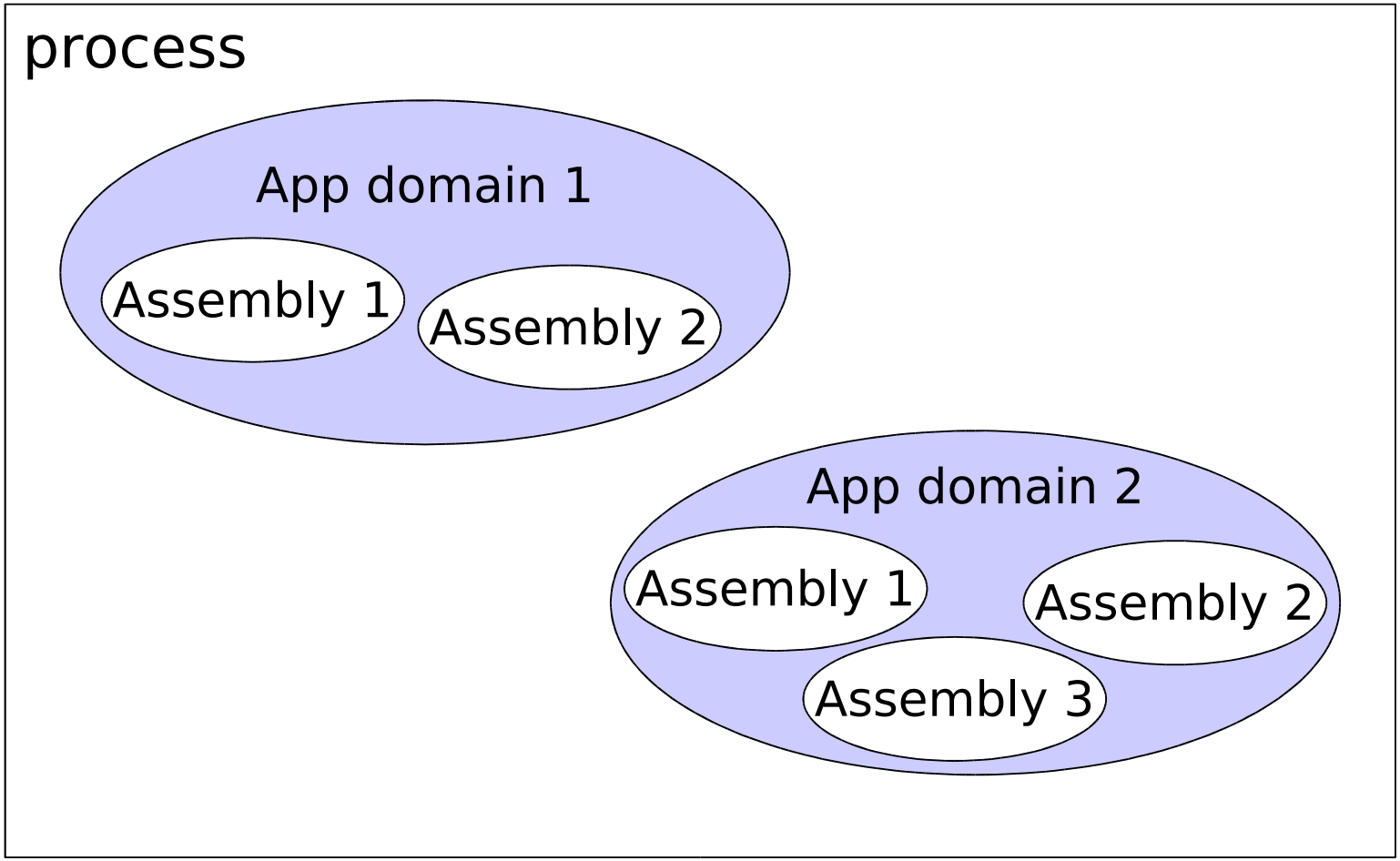
Evidence is dynamically calculated when code is running; e.g. the URL of origin is usually not known in advance.

Evidence associated with assemblies and with application domains (app domains).

App domains: ‘mini-processes’ within processes.

‘Management layer’ above assemblies.

Application Domains



Providing Evidence

Host-provided evidence:

- Host: an unmanaged entity that initiates the CLR (e.g. Internet Explorer) or managed code launching other managed code.
- The kind of evidence mentioned so far.

Assembly provided evidence:

- Provided by an assembly itself.
- Cannot override host-provided evidence.
- Can be any object → application specific access control.
- Custom code needed to process such evidence.

Permissions

Permission: privilege that can be granted to .NET code, e.g. write to file system

- Code access permissions: standard permissions.
- Identity permissions: indicate that an assembly has a certain piece of evidence.
- Other permissions: e.g. PrincipalPermission representing a user identity.

Built-in permissions and permission sets.

Granted by the security policy: takes evidence as input and returns permissions.

Demanded by .NET assemblies: required permissions to access resource.

Declarative & Imperative Sec.

Declarative security actions: stored in the assembly's metadata.

- Can be easily (statically) reviewed on assemblies.
- Occur at the beginning of a method.
- Can be placed at class level.

JIT-time security actions can only be expressed in in declarative form.

Imperative security actions: stored in IL code.

- More complex security logic possible.
- Necessary with dynamic parameters.

Enforcing Policies

Granted permissions of an assembly derived from evidence by evaluating membership conditions.

Code groups and policy levels for managing policy specification.

Enforcement: stack walk, goes through the call stack and checks for required permission.

- No check against the method making the request.

Assert, Deny, PermitOnly: operations that attach permissions to current stack frame; removed when returning from that method.

Modifying the Stack Walk

Assert: terminates stack walk for a given permission granting this permission (all frames examined so far also have the permission.)

- Does not terminate the stack walk if the granted permissions are insufficient for the request.
- Allows “untrusted” callers to call the method successfully.

Deny: terminates stack walk raising an exception.

- Check at run time; mainly useful for testing.
- Do not put the check for the denied permission in the same method as the ‘deny’.

PermitOnly: terminates stack walk raising an exception unless stated permissions are satisfied.

Summary

.NET CLR provides code-identity-based access control.

Stack walk used as the security enforcement algorithm.

Numerous means available for structuring security policies.

- Open question: How to best assign permissions to assemblies?

To use these means in practice you have to study the details of the .NET framework.

Cookies
SPKI
Trust Management
Digital Rights Management

Stateless Protocols

The *http* protocol (hypertext transfer protocol, RFC 1945) is stateless by design.

Even *http* requests coming from the same client are treated as independent events.

- E.g., if a password is required to access a web page, it would have to be returned every time you click on this page.
- Solution in *http* 1.0: browser stores password entered at first request and automatically includes it in all further replies to the server.

Transactions consisting of several steps may need to keep a consistent state between client and server for recovering to a safe state if a communication failure occurs.

Cookies

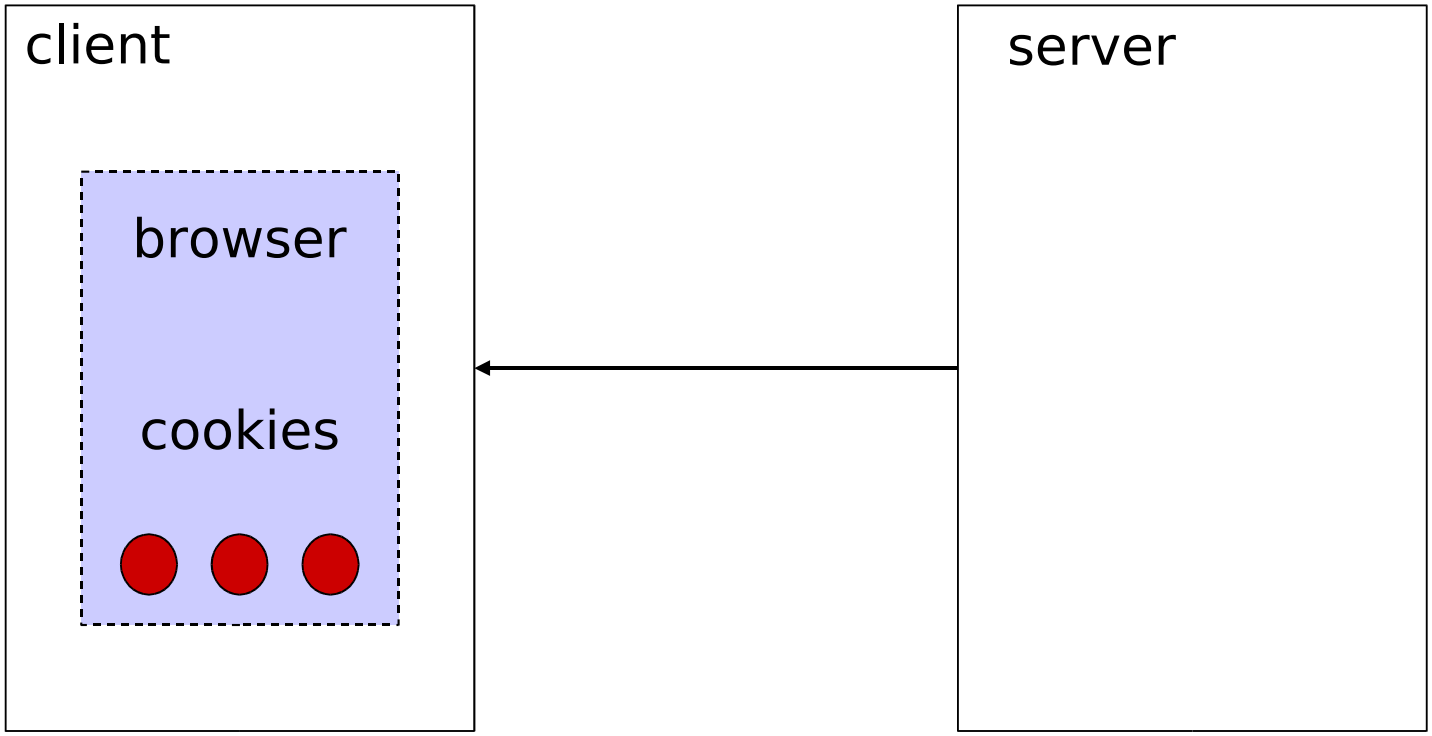
For this reason, the state of the transaction is stored by the browser on the client side in a cookie.

The server can retrieve the cookie to learn the client's current state in the transaction.

With cookies, stateful *http* sessions can be created.

Depending on the duration for which cookies are kept, the concept of a session can be extended beyond a single transaction.

Cookies



Security Issues?

Cookies cannot violate the integrity of your system; they are data, not executable code.

Individual cookies do not disclose information to the server; the server asks the browser to store the cookie.

Usually, cookies are domain specific and servers are only get access to cookies belonging to their domain; in this sense, there is also no new confidentiality.

A server can violate client privacy by creating client profiles, combining information from cookies placed by different servers or by observing client behaviour over time.

Security Issues

A client may change cookies to gain benefits from the server the customer is not entitled to.

- Cookie poisoning attack: assume a server uses the cookie to store bonus points in a loyalty scheme; a client could increase the score to get higher discounts.

'Identity theft': a third party could make an educated guess about a client's cookie and use a spoofed cookie to impersonate the client.

Clients can protect themselves by setting up their browsers to control the placement of cookies:

- Ask for permission before storing a cookie (can easily become a nuisance), block cookies altogether, deleting the cookies at the end of a session.

The server could protect itself by encrypting cookies.

To stop spoofing attacks use proper authentication.

SPKI

Old paradigm: access rules stored locally in protected memory.

Decentralized access control: protect integrity of access rules by cryptographic means; encode rules in digitally signed certificates.

Identity-based access control can be implemented with X.509 identity certificates.

SPKI (Simple Public Key Infrastructure, RFC 2692, 2693): PKI for access control (authorization) that works without user identities.

Local and Global Names

In access control names have essentially only a local meaning within a security domain, and just serve as pointers to access rights.

Interaction between domains: we need to refer to names from other local name spaces; we require globally unique identifiers for name spaces to avoid confusion about names.

Public/private key pairs generated at random are unique with high probability; the public key of an issuer (or its hash) can serve as the unique identifier for the name space defined by that issuer.

Name certificates signed with the private key define a name in the local name space.

Access Rights

Access rights are bound directly to public keys through authorization certificates.

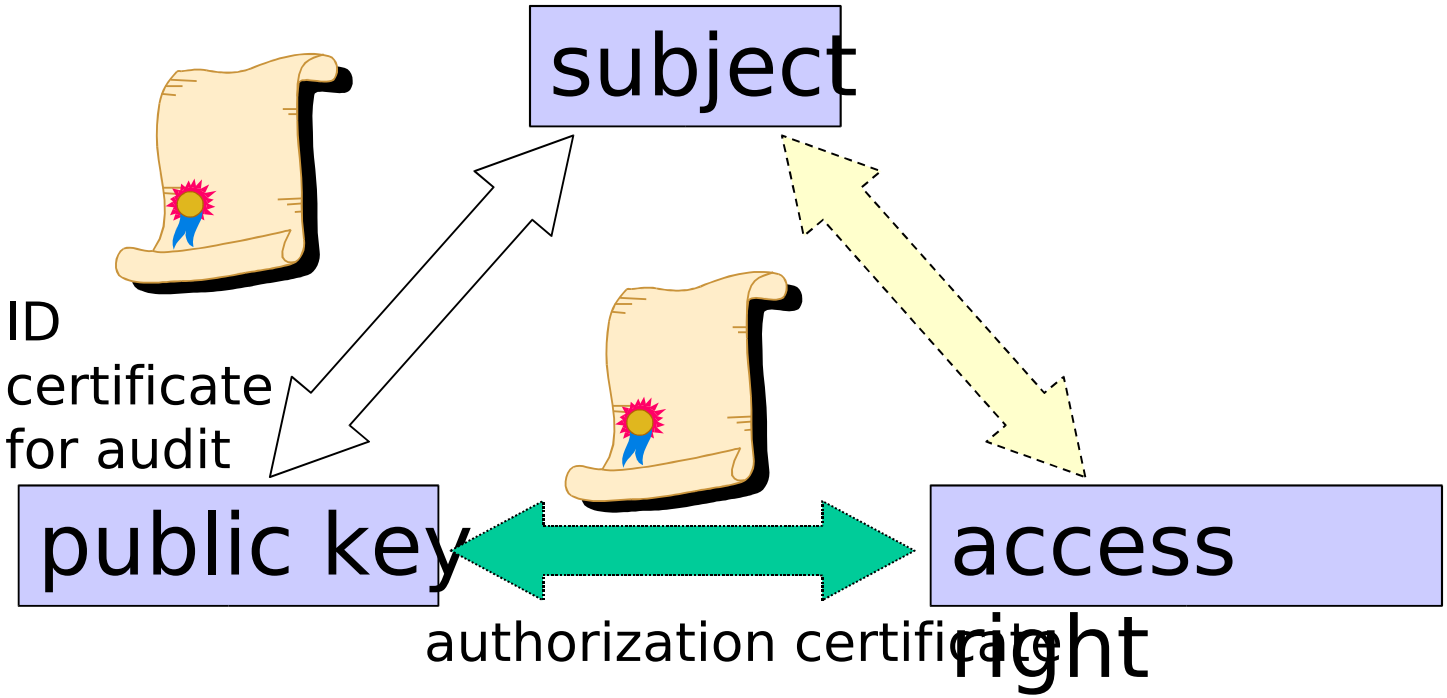
Authorization certificates contain at least an issuer and a subject, and may also include a delegation bit, authorizations, and validity conditions.

The issuer sets policy by signing a certificate and thereby authorizing the subject.

The subject is typically a public key, the hash of a key, or the name for a key.

The root key for verifying certificate chains is stored in an ACL.

SPKI: Access Control



Key-centric access control, ID certificates for accountab

SPKI Policy

Tuples: abstract notation for certificates or ACL

entries:

1. Issuer: public key (or “Self”)
2. Subject: public key, name identifying a public key, hash of an object, …
3. Delegation: TRUE or FALSE
4. Authorization: access rights
5. Validity dates: not-before date and not-after date

Tuple Reduction Algorithms

Evaluates ‘certificate chains’. Authorisations and validity periods can only be reduced.

Input: <Issuer1,Subject1,D1,Auth1,Val1>
<Issuer2,Subject2,D2,Auth2,Val2>

IF Subject1 =Issuer2 AND D1 = TRUE

THEN output <Issuer1, Subject2, D2,
AIntersect(Auth1,Auth2),
VIntersect(Val1,Val2)>

SPKI – Evaluation

SPKI Certificate Theory is recommended reading on names, access control, etc.

Oriented towards access control and away from global CA hierarchies; separation of concerns:

- ID certificates for accountability
- Attribute and authorisation certificates for access control: certificates ≈ distributed storage of ACLs

SPKI standardizes (prescribes?) policy decisions: e.g. only permissions held by delegator can be delegated; does not support separation of duties.

Trust Management

Traditionally, access rules can be found in a well defined place: ACL in a parent directory

Traditionally, a subject presents its credentials and the reference monitor decides on the basis of the input it has received, and does not ask third parties for decisions

In open environments, we frequently encounter situations involving third parties

Example

Service Level Agreement between telecom providers *X* and *Y* that gives customers from *X* access to the services offered by *Y*.

- *Y* will not get a list of all subscribers from *X*.
- *X* issues its subscribers with certificates and gives *Y* the required verification key.
- Subscribers from *X* request services from *Y* by presenting their certificates.
- Provider *Y* calls back *X* to perform an on-line status check on the certificates, 'deferring' this check to *X*.
- The reply from *X* is input to *Y*'s decision.

Trust Management

A unified approach to specifying and interpreting security policies, credentials, and relationships introduced in PolicyMaker

- M. Blaze, J. Feigenbaum, J. Lacy: Decentralized Trust Management, 1996 IEEE Symposium on Security & Privacy.

Generalize rules: instead of ACLs, use a programming language to express assertions.

Assertion: bind a public key to a predicate on actions.

- Authorizes an action if a digitally signed request to perform this action can be verified with the public key given in the assertion and if the action satisfies the predicate.

Trust Management

Credentials can directly authorize actions, there is no need to authenticate a user (like in SPKI).

Distribute authority: assertions can be local ('policies') or be signed by another authority ('credentials').

Trust management engine (compliance checker) answers question:

"Does the set C of credentials prove that the request r complies with the local security policy P?"

Trade-off between expressiveness of the language and complexity of the compliance checker.

Digital Rights Management

Digital Rights Management (DRM): enforce vendor policies on a customer machine.

Departure from ‘old’ access control paradigm:

- Policies enforced on a system are no longer set by the owner but by an external party.
- The adversary is no longer an external party trying to subvert the system but an owner trying to bypass the policy.
- Security goal: integrity of the access control system, as interpreted by the external party.

Trusted Platform Modules could provide ‘truthful’ reports about the hardware and software configuration of a target machine.