Authentication & Authorization

Sean Kelly
Object Oriented Data Technology
2004.11.15

NASA
JET PROPULSION LABORATORY

Overview

- Security Manager
  - Security policy
  - Permissions
  - Access control context
  - Privileged code
- Subjects and Principals
  - Login contexts
  - Callback handlers
  - Principal permissions
- Remote Authentication and Authorization

Java Security Features

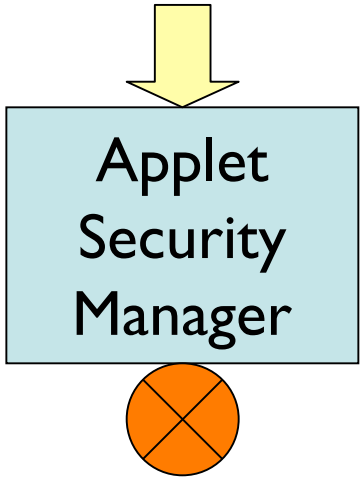
- Permissions and authorization
  - Various specific permissions
  - Policy files
- Access Control
  - Protection domains
  - Privileges
- Secure Class Loading
- Cryptography
  - Signatures
  - Keys and certificates
  - Encryption
- Authentication

Security Manager

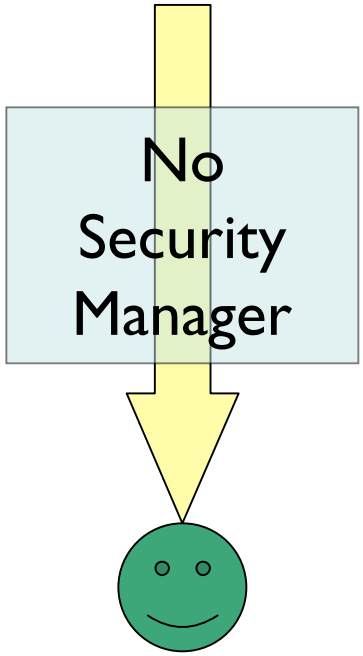
- Default: apps run with no Security Manager
  - Applets run with a Security Manager installed by the browser
- No Security Manager means
  - All activities are allowed
  - Permissions checks are turned off

Security Manager

Read property
"user.home"



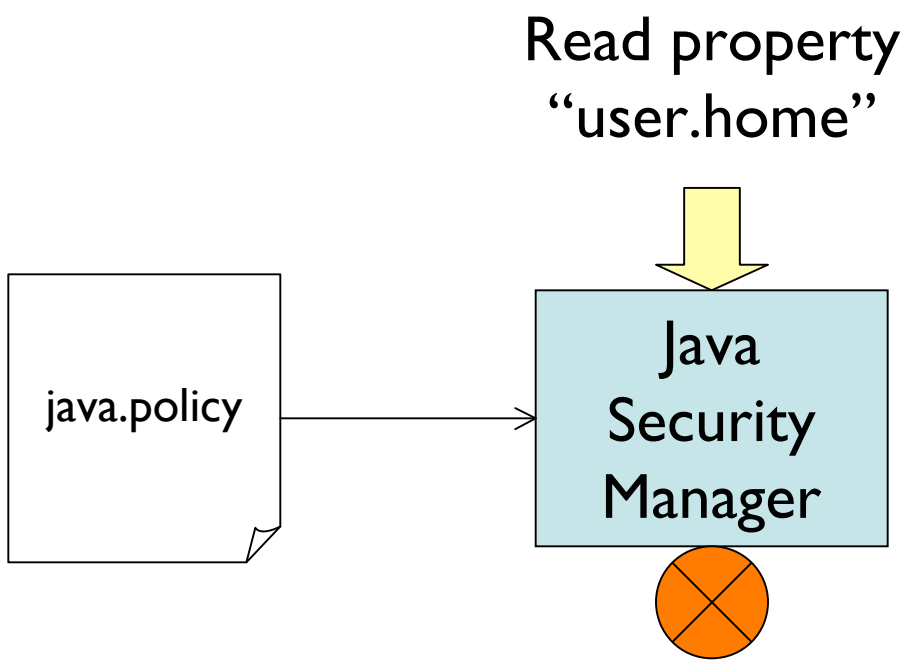
Read property
"user.home"



Applications *Can* Have Security

- Run with the java.security.manager system property:
  `java -Djava.security.manager -jar app.jar`
- Runs with the default Java Security Manager
  - Interface: java.lang.SecurityManager

Application Security



The Security Policy

- See $J2SDK/jre/lib/security/java.policy
- This is the policy file used by default
  - You can specify your own policy file
  - Or even your own SecurityManager
    - Doesn't have to use policy files at all
      - Hard code the policy
      - Read it from a database
      - Etc.

Create a Policy File

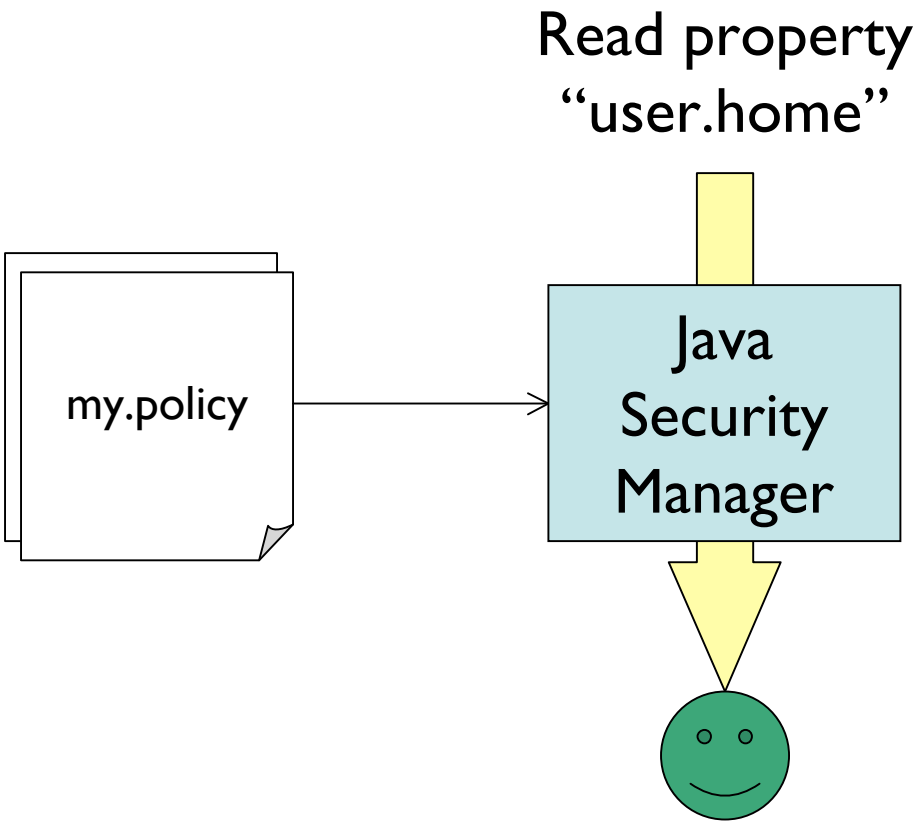
- It's plain text, say a file "my.policy":

```
grant codeBase "file:${user.home}/app.jar" {
    permission java.util.PropertyPermission
        "user.home", "read";
};
```

- Specify it when running:

```
java -Djava.security.manager
    -Djava.security.policy=my.policy
    -jar app.jar
```

Custom Security



Granting Permissions

- You grant permissions to
  - Code bases
    - Jars
    - Directories of classes
  - Signed code bases (signed jars)
  - Principals
    - Users, corporations, roles
    - Other entities

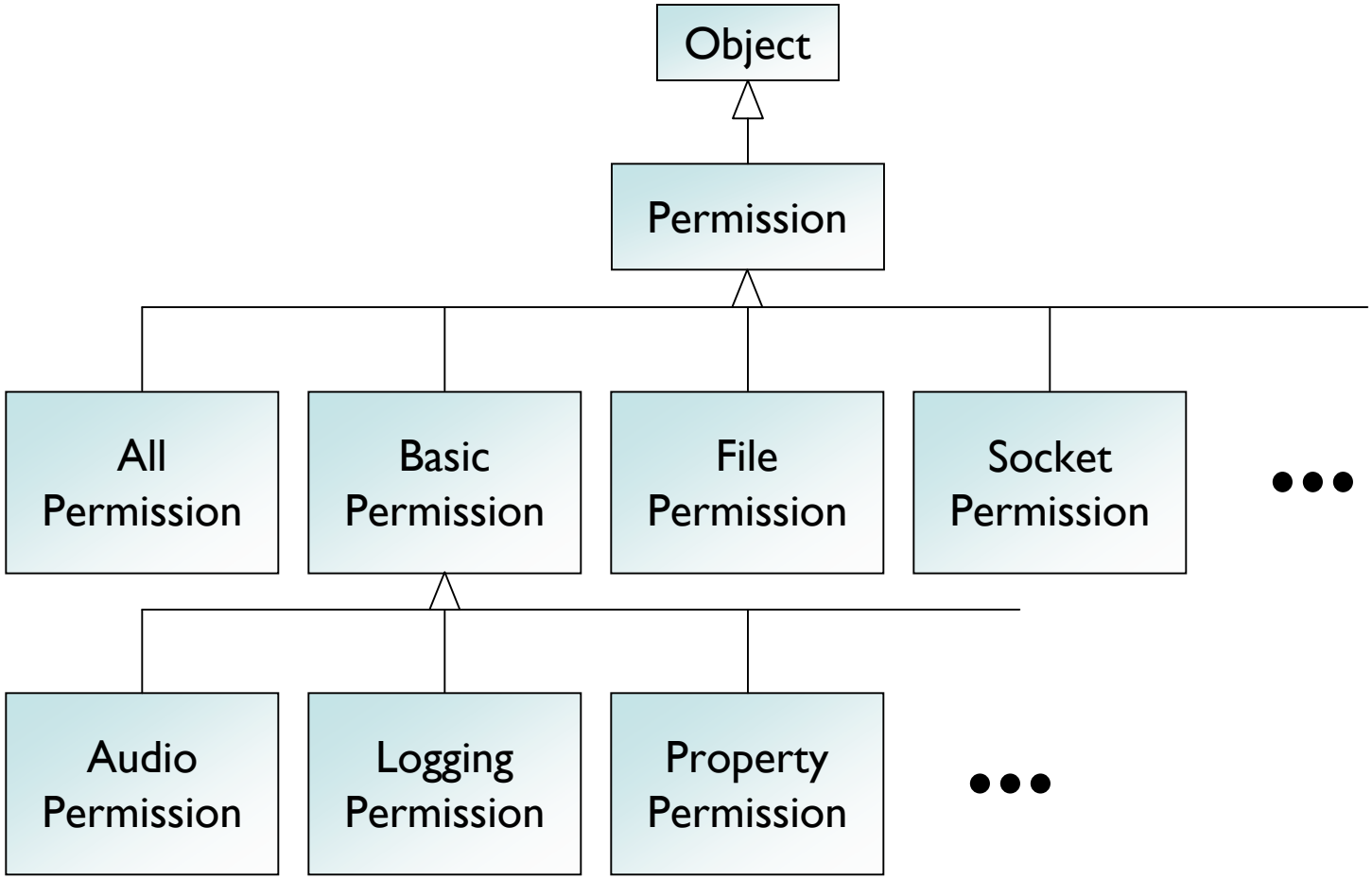
Grant Syntax

- To a code base
  grant codeBase “http://oodt/jars/profile.jar” {...
- To a signed jar
  keystore “${user.home}/my.ks”;
  grant signedBy “Dan” {...
- To a principal
  grant principal jpl.pds.User “Steve” {...

Granting Permissions

- Permissions are objects
  - Classes extend java.security.Permission
  - Define your own

Permissions



Permission Implication

- One permission can imply another
  - Suppose *a* implies *b*
  - Then if *a* is granted, *b* is implicitly granted

Permission Targets

- Permissions always have a String target
  - java.lang.RuntimePermission
    - “createClassLoader”
    - “exitVM”
    - Etc.
  - java.awt.AWTPermission
    - “accessClipboard”
    - “accessEventQueue”
    - Etc.
  - Etc.

Permission Actions

- Some Permissions have targets *and* actions
  - java.io.FilePermission
    - "/tmp/*", "read,write"
    - "/Applications/-", "read,execute"
    - Etc.
  - java.net.SocketPermission
    - "*.jpl.nasa.gov", "connect"
    - "localhost:1024-", "accept,connect,listen"
    - Etc.
  - Etc.

Granting Permissions

- Suppose a distributed application needs to connect to a network port:

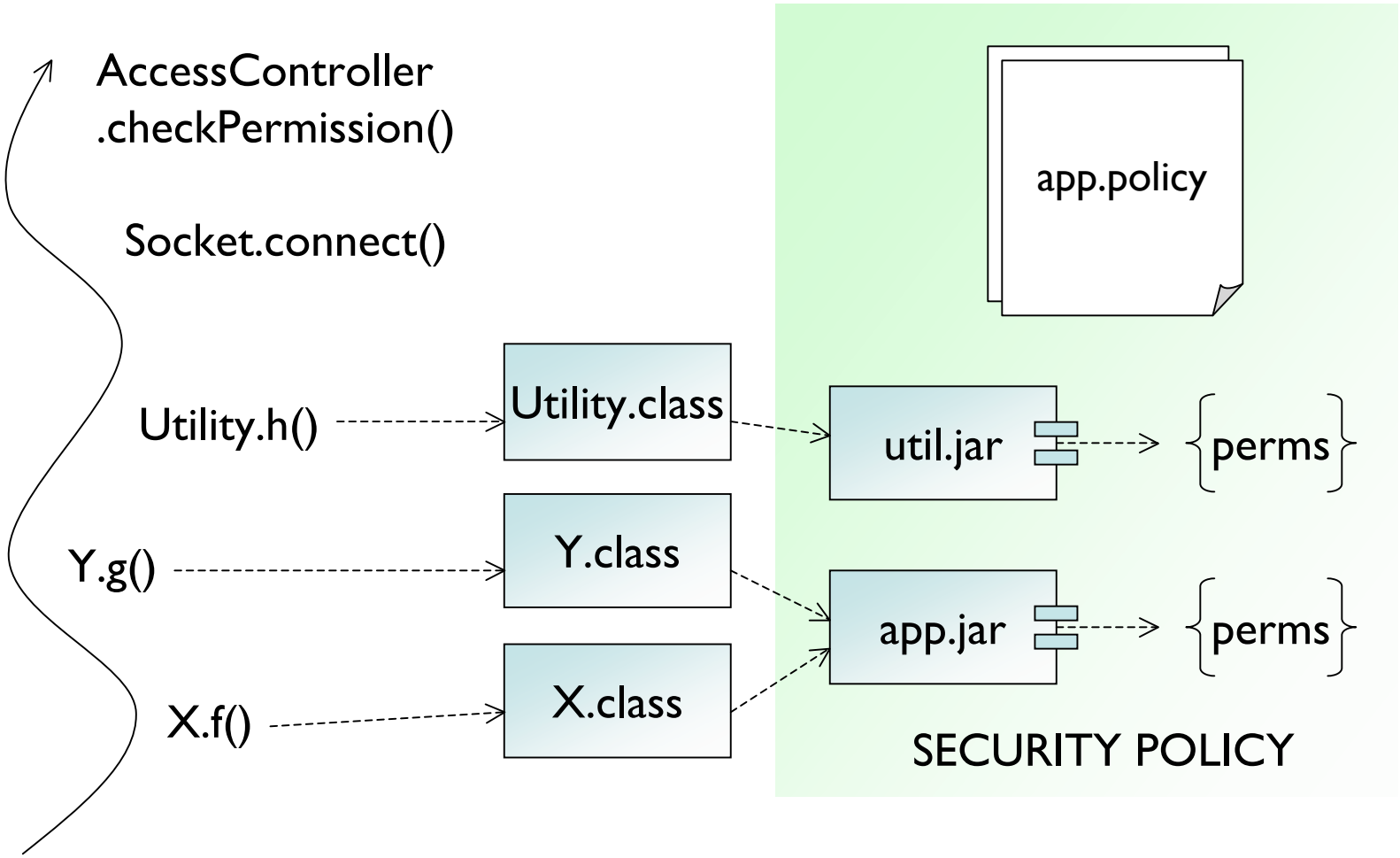
```
grant codeBase ??? {
    permission java.net.SocketPermission \
            "dmie.jpl.nasa.gov:1199", "connect";
};
```

- What code base should receive the permission?

Permission Checks

- Permissions checked by AccessController
  - AccessController checks permissions in an *access control context*
- Access control context
  - Computation's call stack (thread)
  - Protection domains (code bases, signatures)
  - Permissions granted (policy files)

Access Control Context



Checking Permissions

- Somewhere in Socket.java:

```
SocketPermission p =
   new SocketPermission(host, port);
AccessController.checkPermission(p);
```

  - Where `host` is passed in, "dmie.jpl.nasa.gov"
  - And `port` is passed in, 1199

Permission Checks

- If any caller in the chain lacks permission
  - Throw AccessControlException
  - *Unless*
    - A caller whose domain is granted the permission is marked as privileged
      - All subsequent callers are considered privileged

Marking Code as Privileged

- Enables a piece of code to have more access to resources
  - *More than the code that calls it*
- A permission check can succeed when called by privileged code
- Unix analogue: "set-uid bit"
  - You can't write the /etc/passwd file
  - But you need to change your password
  - Run the "/bin/passwd" program
    - Has the set-uid bit, runs with root privileges

Marking Code as Privileged

- Somewhere in Y.java:

```
void g() {
   AccessController.doPrivileged(
      new PrivilegedAction() {
         public Object run() {
            Utility.h("dmie.jpl.nasa.gov", 1199);
         }
      }
   );
```

What About Users?

- Use Subject.doAsPrivileged
  - Instead of AccessController.doPrivileged
- Uses permissions granted to *principals*
  - Instead of to code bases or signed code

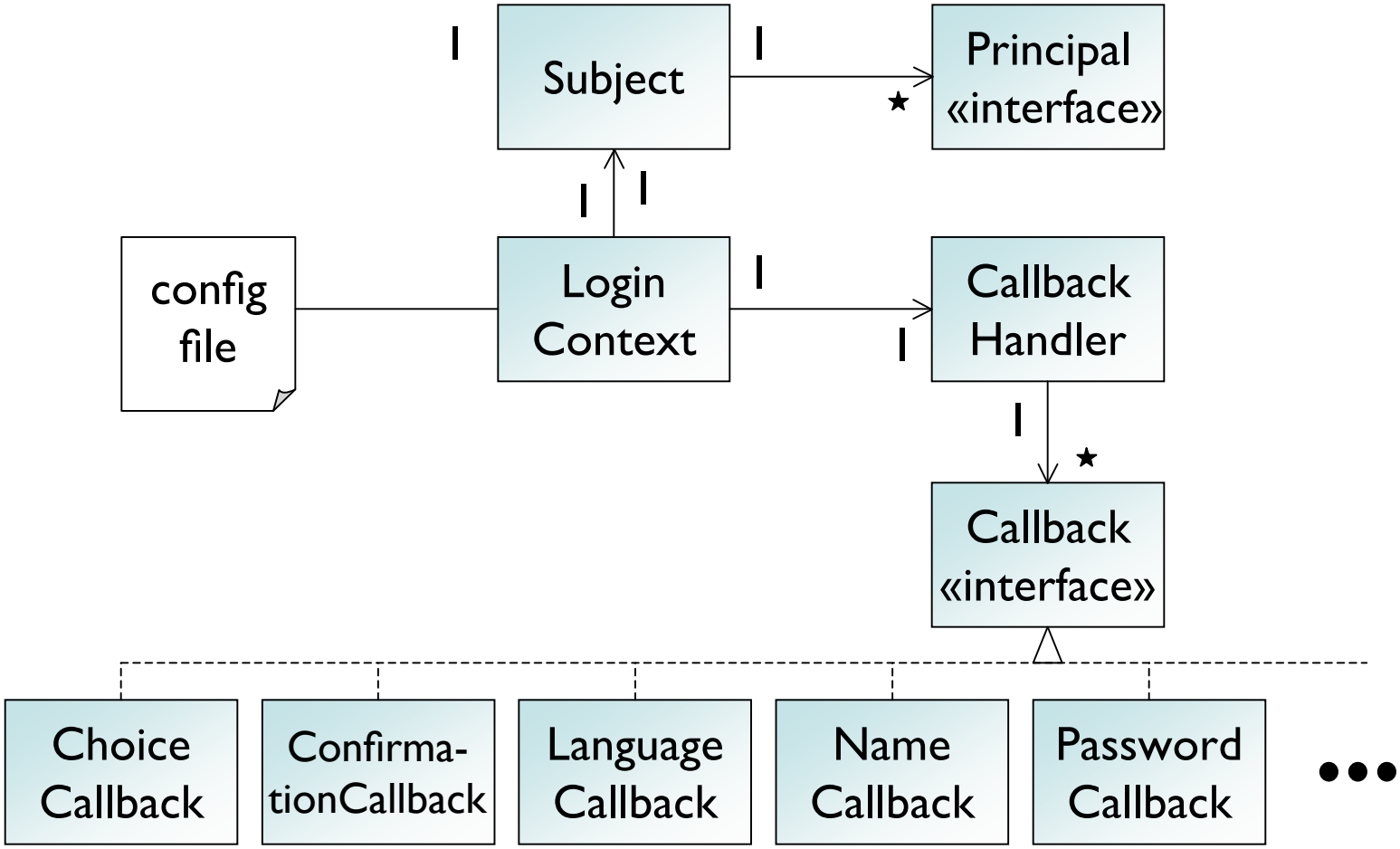
Principals?

- Principals are
  - *Authentic* aspects of a Subject
  - SSN, login name, job title, mission role, etc.
- Subjects?
  - *Entities* that make requests
  - People, corporations, services, etc.

Authenticating

- Java doesn't know what you're logging into
  - Or in what context
    - Text program
    - Web app
    - GUI
  - So it provides a *framework:* JAAS
- Framework parts
  - Configuration file
  - Callbacks for credentials
  - Principal interface, Subject class

Framework


Subject
Principal
«interface»
config
file
Login
Context
Callback
Handler
Callback
«interface»
Choice
Callback
Confirma-
tionCallback
Language
Callback
Name
Callback
Password
Callback
1
1
*
1
1
1
1
1
*

Configuration File

- Describes login contexts
  - Name
  - Login modules
  - Module obligation
    - Required
    - Requisite
    - Sufficient
    - Optional
  - Flags

Example Configuration

- Say, "/usr/local/pds/etc/pds.auth":

```
archive {
 jpl.pds.PDSAdmins sufficient;
 jpl.pds.PDSUsers required;
};
products {
 jpl.pds.PDSUsers sufficient;
 com.sun.security.auth.module.UnixLoginModule required;
};
```

Logging In

- Create the LoginContext:
  ```
  LoginContext ctx = new LoginContext("archive",
     new MyCallbackHandler());
  ```
- Call the login method:
  ```
  ctx.login();
  ```
  – Throws LoginException if user cannot authenticate
- Get the subject's authentic principals:
  ```
  Set principals = ctx.getSubject().getPrincipals();
  ```

CallbackHandler

- Handles requests from a LoginModule for user interaction
  - Display prompt
  - Get user name
  - Get password
  - Etc.
- Single method:
  `void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException;`

MyCallbackHandler

- One possible implementation:

```
public void handle(Callback[] cbs) {
  for (int i = 0; i < cbs.length; ++i) {
    if (cbs[i] instanceof NameCallback) {
      NameCallback n = (NameCallback) cbs[i];
      System.out.println(n.getPrompt());
      n.setName(readName());
  } else if (cbs[i] instanceof PasswordCallback) {
      PasswodCallback p = (PasswordCallback) cbs[i];
      System.out.println(p.getPrompt());
      p.setPassword(readPassword());
  } ...
```

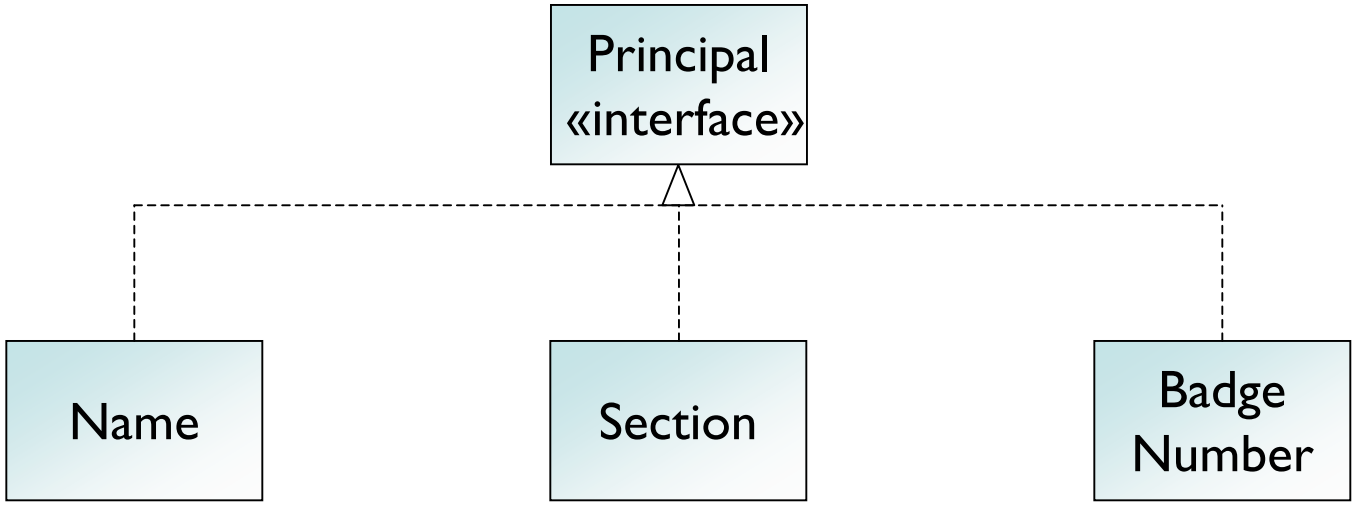
Principals

- java.security.Principal
  - A simple interface:

```
public interface Principal {
    String getName();
    boolean equals(Object rhs);
    int hashCode();
    String toString();
}
```

- LoginModules insert *authentic* Principals into a Subject

Potential App-Specific Principals



Granting Principal Permissions

- Into the policy file:

```
grant principal jpl.security.Name “Crichton” {
  permission nbs.AcctPerm “budget”, “modify”;
};
grant principal jpl.security.Section “389” {
  permission oao.CapitalPerm “Mac”, “upgrade-desktop”;
  permission pds.ArchivePerm “ingest”;
};
grant principal jpl.security.Badge “104443” {
  permission nbs.AcctPerm “budget”, “read”;
  permission pds.ArchivePerm “ingest,retrieve”;
  permission pds.ProductPerm “query”;
};
```

Checking Principal Permissions

- Permissions checks fail if
  - A caller in the access control context lacks permission
  - *Unless* a caller is marked as privileged
    - And the caller is assuming the principal identity

Example: Ingest Data

- Caller:

```
LoginContext ctx = new LoginContext("archive",
   new MyCallbackHandler();
ctx.login();
Subject subject = ctx.getSubject();
Subject.doAsPrivileged(subject,
  new PrivilegedAction() {
    public Object run() {
      ingest(someData);
    }
  }, /*accessControlContext*/null);
```

Example: Ingest Data

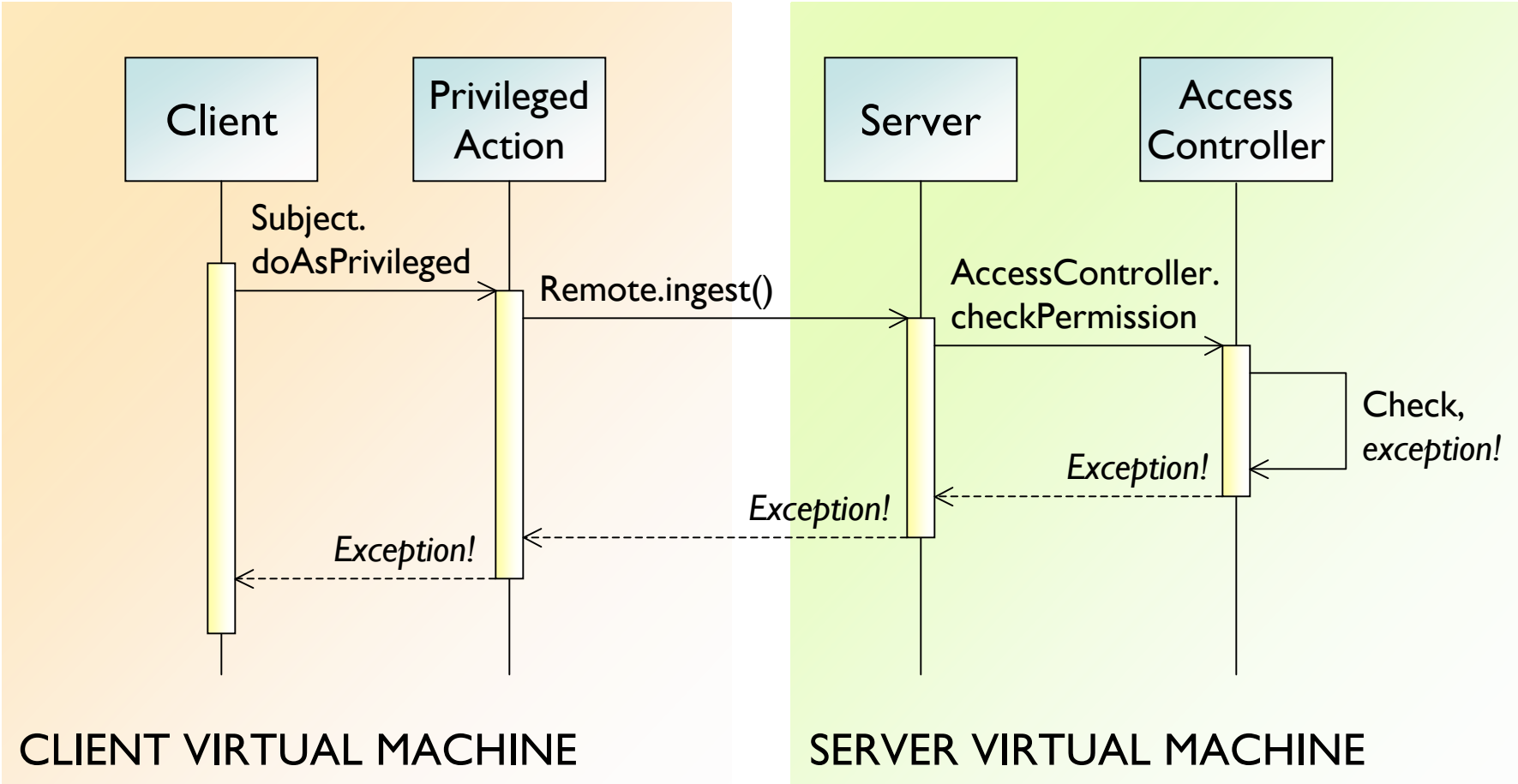
- Called:

```
public void ingest(byte[] someData) {
    ArchivePerm ap = new ArchivePerm("ingest");
    AccessController.checkPermission(ap);
    save(someData);
}
```

Remote Methods

- What if `ingest` is a remote method?

Remote Permission Checks



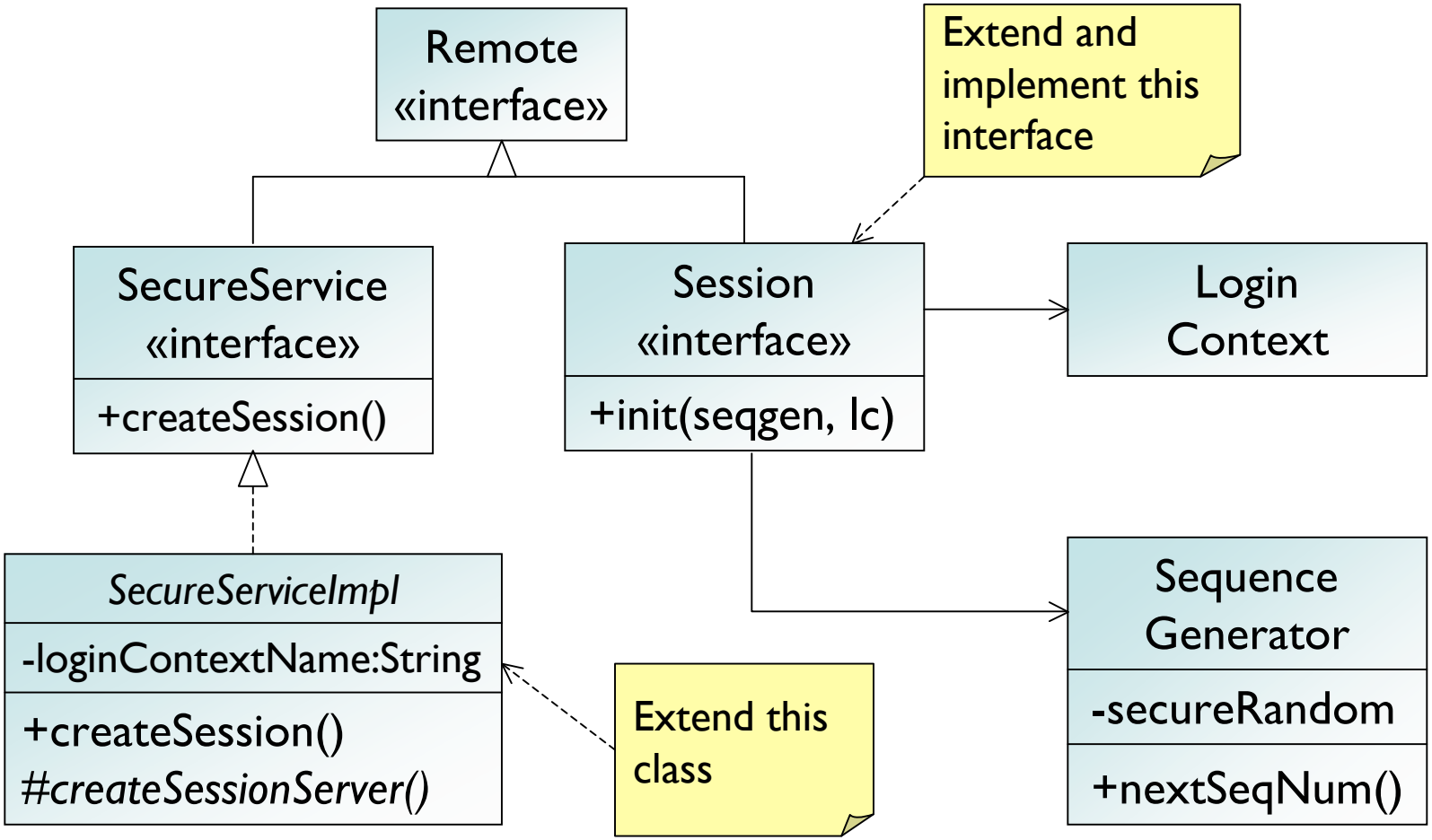
It Doesn't Work

- Server's access control context
  - Has no privileged callers
  - Privilege is *not* carried across remote calls
- That's *good!*
  - Sniffing the network and producing the same data could grant permission to an attacker
- Solution?
  - Authenticate subjects *in the server with remote callbacks to the client*

Remote Auth & Auth

- Use edm-security-1.0.2 or later
  - Framework for remote services
    - That are authenticated
    - And authorized

Server Side Authentication



Classes and Interfaces

- SecureServiceImpl
  - Abstract class
  - Super-server for auth/auth remote services
  - You implement createSessionServer()
    - Returns an object that implements Session
- Session
  - Interface
  - Represents session with a single authenticated user
  - You implement init()
  - Add additional app-specific methods

Example: an Archive Service

- The super-server:

```
public class ArchiveService extends
  SecureServiceImpl {
  public ArchiveService()
    throws RemoteException {
    super("archive"); // "archive" is login context name
  }
  protected Session createSessionServer()
    throws RemoteException {
    return new ArchiveSessionImpl();
  }
}
```

Example: an Archive Service

- The session interface:

```
public interface ArchiveSession extends Session{
  void ingest(int seq, byte[] data);
}
```

Example: an Archive Service

- The session implementation:

```
public class ArchiveSessionImpl
  implements ArchiveSession {
  public ArchiveSessionImpl()
    throws RemoteException { super(); }
  public synchronized void init(
    SequenceGenerator sg, LoginContext lc
  ) {
    this.sg = sg; this.lc = lc; subject = lc.getSubject();
  }
```

Example: an Archive Service

- Continued…

```
void ingest(int seq, byte[] data)
  throws Exception {
  checkLogin(seq);
  Subject.doAsPrivileged(subject,
    new PrivilegedAction() {
      public Object run() {
        ingest(data);
      }
    }, /*acc*/null);
}
```

Example: an Archive Service

- Continued…

```
private void ingest(byte[] data) {
  ArchivePermission p =
    new ArchivePermission("ingest");
  AccessController.checkPermission("ingest");
  // if we get here, we can save  the data ...
  saveTheDataSomeHow(data);
}
```

Example: an Archive Service

- Concluded…

```
 private void checkLogin(int seq) {
  if (lc == null)
   throw new Exception("logged out");
  if (seq == sg.nextSeqNum()) return;
  try {
   lc.logout();
  } catch (LoginException ignore) { }
  lc = null;
  throw new Exception("invalid seq num");
 }
 private SequenceGenerator sg;
 private LoginContext lc;
}
```

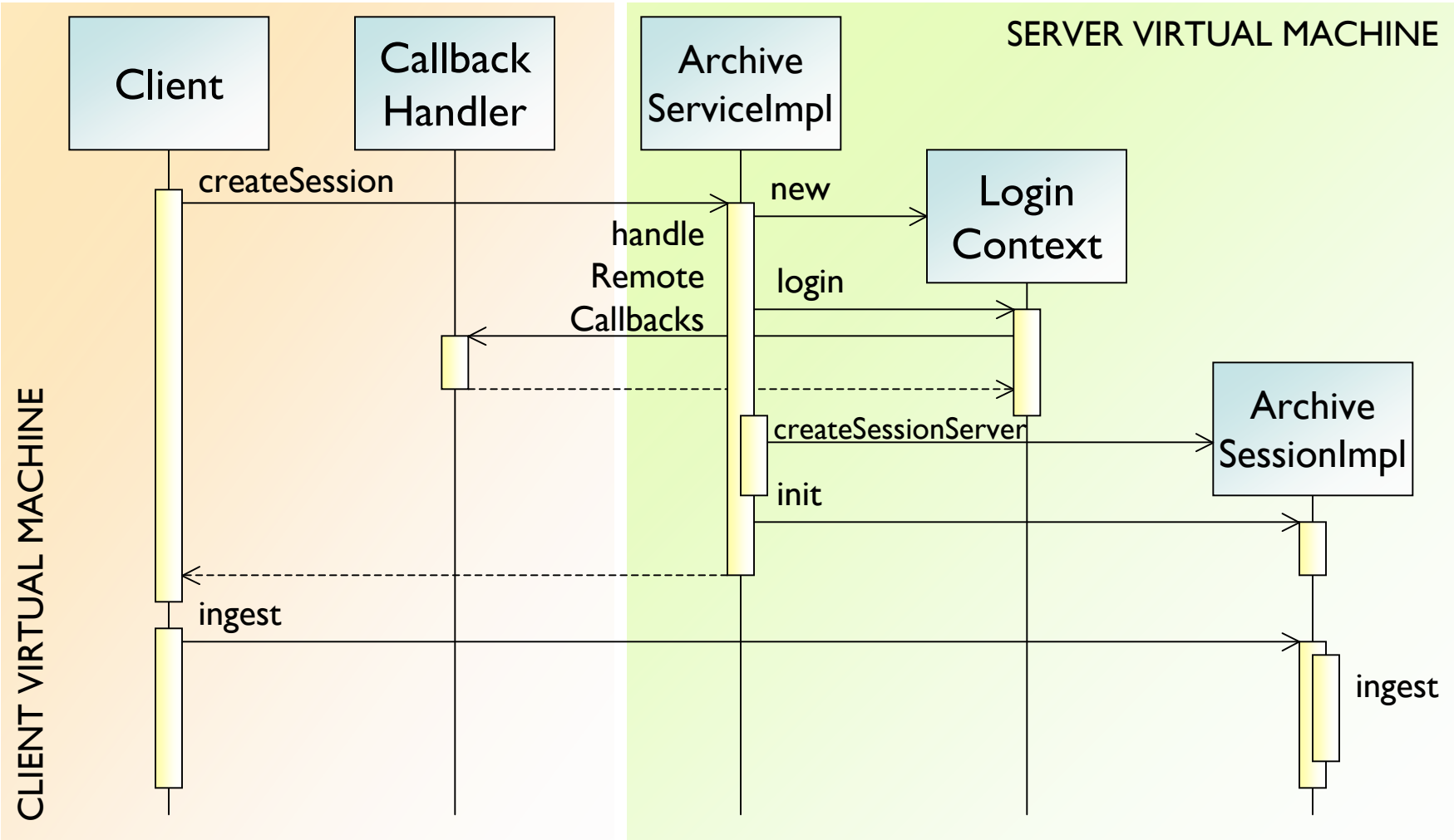
How to Invoke It

- Client finds reference to ArchiveService
  - Calls createSession(…), passing in seed and a RemoteCallbackHandler
    - Seed is for secure sequence generator
    - RemoteCallbackHandler is like CallbackHandler
      - But for RemoteCallbacks
  - Service attempts to login
    - Calls RemoteCallbacks to authenticate credentials
- If successful, client gets reference to ArchiveSession
  - It must pass the next sequence number in on every operation

Sequence Numbers?

- Generated from a secure pseudo-random number generator
  - SHA1PRNG
- Prevents attacker from hijacking the next operation
- Client chooses the seed to PRNG
  - Passes seed to super-server in createSession
  - Super-server communication must be over SSL/TLS
  - Session communication can occur in the clear

Interaction Timeline



Summary

- Security Manager
  - Security policy
  - Permissions
  - Access control context
  - Privileged code
- Subjects and Principals
  - Login contexts
  - Callback handlers
  - Principal permissions
- Remote Authentication and Authorization

Exercises

1. Add authorization and authentication to existing OODT services (show your work):
   - Profile service
   - Archive service
   - Metadata service
2. Implement a login context to your own discipline's user or role database
3. Implement a remote callback handler that handles user interaction on a web page