Announcements

- HW 5 due on Friday.

- Midterm 2 will be held on Monday
  - I will be out of town

- Agenda:
  - Finish up classical Access Control Matrices
  - Begin moving towards software security:
    Java and C# applet access control models
### Access Control Matrices

<table>
<thead>
<tr>
<th>A[s][o]</th>
<th>Obj₁</th>
<th>Obj₂</th>
<th>…</th>
<th>Objₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subj₁</td>
<td>{r,w,x}</td>
<td>{r,w}</td>
<td>…</td>
<td>{}</td>
</tr>
<tr>
<td>Subj₂</td>
<td>{w,x}</td>
<td>{}</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>Subjₘ</td>
<td>{x}</td>
<td>{r,w,x}</td>
<td>…</td>
<td>{r,w,x}</td>
</tr>
</tbody>
</table>

Each entry contains a set of rights.
Storing the Access Control Matrix

- Subjects >> # users
  - Processes
- Objects >> # files
  - Potentially could have permissions on any resource

- The matrix is typically sparse
  - Store only non-empty entries
Access Control Lists

<table>
<thead>
<tr>
<th>A[s][o]</th>
<th>Obj₁</th>
<th>Obj₂</th>
<th>...</th>
<th>Objₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subj₁</td>
<td>{r,w,x}</td>
<td>{r,w}</td>
<td>...</td>
<td>{}</td>
</tr>
<tr>
<td>Subj₂</td>
<td>{w,x}</td>
<td>{}</td>
<td>...</td>
<td>{r}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Subjₘ</td>
<td>{x}</td>
<td>{r,w,x}</td>
<td>...</td>
<td>{r,w,x}</td>
</tr>
</tbody>
</table>

For each object, store a list of (Subject x Rights) pairs.
Access Control Lists

• Resolving queries is linear in length of the list
• Revocation w.r.t. a single object is easy
• “Who can access this object?” is easy
  – Useful for auditing
• Lists could be long
  – Factor into groups (lists of subjects)
  – Give permissions based on group
  – Introduces consistency question w.r.t. groups
• Authentication critical
  – When does it take place? Every access would be expensive.
### Capabilities Lists

For each subject, store a list of (Object x Rights) pairs.

<table>
<thead>
<tr>
<th>A[s][o]</th>
<th>Obj₁</th>
<th>Obj₂</th>
<th>...</th>
<th>Objₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subj₁</td>
<td>{r,w,x}</td>
<td>{r,w}</td>
<td>...</td>
<td>{}</td>
</tr>
<tr>
<td>Subj₂</td>
<td>{w,x}</td>
<td>{}</td>
<td>...</td>
<td>{r}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Subjₘ</td>
<td>{x}</td>
<td>{r,w,x}</td>
<td>...</td>
<td>{r,w,x}</td>
</tr>
</tbody>
</table>
Capabilities

- A capability is a (Object, Rights) pair
  - Often has an expiration date
  - Used like a movie ticket e.g.:
    (“The Incredibles”, {admit one}, 7:00pm show)

- Should be unforgeable
  - Otherwise, subjects could get illegal access

- Authentication takes place when the capabilities are granted (not needed at use)

- Harder to do revocation (must find all tickets)

- Easy to audit a subject, hard to audit an object
Implementing Capabilities

• Must be able to name objects
• Unique identifiers
  – Must keep map of UIDs to objects
  – Must protect integrity of the map
  – Extra level of indirection to use the object
  – Generating UIDs can be difficult
• Pointers
  – Name changes when the object moves
  – Remote pointers in distributed setting
  – Aliasing possible
Unforgeability of Capabilities

• Special hardware: tagged words in memory
  – Can’t copy/modify tagged words
• Store the capabilities in protected address space
• Could use static scoping mechanism of safe programming languages.
  – Java’s “private” fields
• Could use cryptographic techniques
  – OS kernel could sign (Object, Rights) pairs using a private key
  – Any process can verify the capability
Mobile Code

• Modern languages like Java and C# have been designed for Internet applications and extensible systems

  applet  applet  applet

  VM

  web browser

  operating system

• PDAs, Cell Phones, Smart Cards, …
Applet Security Problems

• Protect OS & other valuable resources.
• Applets should not:
  – crash browser or OS
  – execute “rm –rf /”
  – be able to exhaust resources
• Applets should:
  – be able to access some system resources (e.g. to display a picture)
  – be isolated from each other
• Principles of least privileges and complete mediation apply
Java and C# Security

- Static Type Systems
  - Memory safety and jump safety
- Run-time checks for
  - Array index bounds
  - Downcasts
  - *Access controls*
- Virtual Machine / JIT compilation
  - Bytecode verification
  - Enforces encapsulation boundaries (e.g. private field)
- Garbage Collected
  - Eliminates memory management errors
- Library support
  - Cryptography, authentication, …
Access Control for Applets

• What level of granularity?
  – Applets can touch some parts of the file system but not others
  – Applets can make network connections to some locations but not others

• Different code has different levels of trustworthiness
  – www.l33t-hax0rs.com vs. www.java.sun.com

• Trusted code can call untrusted code
  – e.g. to ask an applet to repaint its window

• Untrusted code can call trusted code
  – e.g. the paint routine may load a font

• How is the access control policy specified?
Java Security Model

VM Runtime

- a.class
- b.class
- c.class
- d.class
- e.class

ClassLoader
SecurityManager

Security Policy

- Domain A
  - Permissions

- Domain B
  - Permissions

http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-specTOC.fm.html
Kinds of Permissions

• `java.security.Permission` Class

```java
perm = new java.io.FilePermission("/tmp/abc","read");

java.security.AllPermission
java.security.SecurityPermission
java.security.UnresolvedPermission
java.awt.AWTPermission
java.io.FilePermission
java.io.SerializablePermission
java.lang.reflect.ReflectPermission
java.lang.RuntimePermission
java.net.NetPermission
java.net.SocketPermission
...
```
Code Trustworthiness

• How does one decide what protection domain the code is in?
  – Source (e.g. local or applet)
  – Digital signatures
  – C# calls this “evidence based”

• How does one decide what permissions a protection domain has?
  – Configurable – administrator file or command line

• Enforced by the classloader
Classloader Resolution

• When loading the first class of an application, a new instance of the URLClassLoader is used.
• When loading the first class of an applet, a new instance of the AppletClassLoader is used.
• When java.lang.Class.forName is directly called, the primordial class loader is used.
• If the request to load a class is triggered by a reference to it from an existing class, the class loader for the existing class is asked to load the class.

• Exceptions and special cases… (e.g. web browser may reuse applet loader)
Example Java Policy

grant codeBase "http://www.l33t-hax0rz.com/*" {
    permission java.io.FilePermission("/tmp/*", "read,write");
}

grant codeBase "file://$JAVA_HOME/lib/ext/*" {
    permission java.security.AllPermission;
}

grant signedBy "trusted-company.com" {
    permission java.net.SocketPermission(...);
    permission java.io.FilePermission("/tmp/*", "read,write");
    ...
}

Policy information stored in:
$JAVA_HOME/lib/security/java.policy
$USER_HOME/.java.policy
(or passed on command line)
Example Trusted Code

Code in the System protection domain

```java
void fileWrite(String filename, String s) {
    SecurityManager sm = System.getSecurityManager();
    if (sm != null) {
        FilePermission fp = new FilePermission(filename, "write");
        sm.checkPermission(fp);
        /* … write s to file filename (native code) … */
    } else {
        throw new SecurityException();
    }
}

public static void main(...) {
    SecurityManager sm = System.getSecurityManager();
    FilePermission fp = new FilePermission("/tmp/*", "write,...");
    sm.enablePrivilege(fp);
    UntrustedApplet.run();
}
```
Example Client

Applet code obtained from http://www.l33t-hax0rz.com/

class UntrustedApplet {
    void run() {
        ...
        s.FileWrite("/tmp/foo.txt", "Hello!");
        ...
        s/FileWrite("/home/stevez/important.tex", "kwijibo");
        ...
    }
}
Stack Inspection

• Stack frames are annotated with their protection domains and any enabled privileges.

• During inspection, stack frames are searched from most to least recent:
  – fail if a frame belonging to a domain not authorized for privilege is encountered
  – succeed if activated privilege is found to be enabled in the frame
main(...){
    fp = new FilePermission("/tmp/*","write,...");
    sm.enablePrivilege(fp);
    UntrustedApplet.run();
}
Stack Inspection Example

```java
main(...){
    fp = new FilePermission("/tmp/*","write,...");
    sm.enablePrivilege(fp);
    UntrustedApplet.run();
}
```
void run() {
    ...
    s.FileWriter("/tmp/foo.txt", "Hello!");
    ...
}

main(...){
    fp = new FilePermission("/tmp/*","write,...");
    sm.enablePrivilege(fp);
    UntrustedApplet.run();
}
Stack Inspection Example

```java
void fileWrite("/tmp/foo.txt", "Hello!") {
    fp = new FilePermission("/tmp/foo.txt", "write");
    sm.checkPermission(fp);
    /* ... write s to file filename ... */
}

void run() {
    ...
    s.FileWrite("/tmp/foo.txt", "Hello!");
    ...
}

main(...){
    fp = new FilePermission("/tmp/*", "write,...");
    sm.enablePrivilege(fp);
    UntrustedApplet.run();
}
```
Stack Inspection Example

```java
void fileWrite("/tmp/foo.txt", "Hello!") {
    fp = new FilePermission("/tmp/foo.txt","write")
    sm.checkPermission(fp);
    /* ... write s to file filename ... */
}

void run() {
    ...
    s.FileWrite("/tmp/foo.txt", "Hello!");
    ...
}

main(...){
    fp = new FilePermission("/tmp/*","write,...
    sm.enablePrivilege(fp);
    UntrustedApplet.run();
}
```
Stack Inspection Example

```java
void run() {
    ...
    s.FileWrite("/home/stevez/important.tex", "kwijibo");
}

main(...){
    fp = new FilePermission("/tmp/*","write,...");
    sm.enablePrivilege(fp);
    UntrustedApplet.run();
}
```

Policy Database
Stack Inspection Example

```java
main(...){
    fp = new FilePermission("/tmp/*","write,...");
    sm.enablePrivilege(fp);
    UntrustedApplet.run();
}

void fileWrite(".../important.txt","kwijibo") {
    fp = new FilePermission("important.txt","write");
    sm.checkPermission(fp);
}

void run() {
    ... 
    s.FileWrite("/home/stevez/important.tex","kwijibo");
}

CSE331 Fall 2004

Fail

Policy Database

fp
```
Other Possibilities

• The `fileWrite` method could enable the write permission itself
  – Potentially dangerous, should not base the file to write on data from the applet
  – … but no enforcement in Java (information flow would help here)

• A trusted piece of code could `disable` a previously granted permission
  – Terminate the stack inspection early
checkPermission(T) {
    // loop newest to oldest stack frame
    foreach stackFrame {
        if (local policy forbids access to T by class executing in
            stack frame) throw ForbiddenException;

        if (stackFrame has enabled privilege for T)
            return;  // allow access

        if (stackFrame has disabled privilege for T)
            throw ForbiddenException;
    }

    // end of stack
    if (Netscape || ...) throw ForbiddenException;
    if (MS IE4.0 || JDK 1.2 || ...) return;
}