Access Control

- Last time: Unix/Windows access control at the OS level.
  - Access control matrix
  - Reference monitor

- Today: Stack Inspection

- What are the security issues in mobile code?
Mobile Code

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

- PDAs, Cell Phones, Smart Cards, …
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, …
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
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?
• How is it enforced?
Java Security Model

VM Runtime

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

Security Policy

- Domain A
- 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 Hierarchy

- Primordial ClassLoader
- ClassLoader
  - SecureClassLoader
  - URLClassLoader
    - AppletClassLoader
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

```java
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 someone not authorized for privilege is encountered
  - **succeed** if activated privilege is found in frame
Stack Inspection Example

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

Policy Database
Stack Inspection Example

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

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

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

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

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

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

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

Policy Database

Fail

fp
```

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

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

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

- The fileWrite method could enable the write permission itself
  - Potentially dangerous, should not base which file to write on data provided by 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
Stack Inspection Algorithm

cHECK_PERMISSION(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 || ...) return;
}
Two Implementations

• On demand –
  – On a checkPermission invocation, actually crawl down the stack, checking on the way
  – Used in practice

• Eagerly –
  – Keep track of the current set of available permissions during execution (security-passing style Wallach & Felten)
  + more apparent (could print current perms.)
  - more expensive (checkPermission occurs infrequently)
Stack Inspection

- Stack inspection seems appealing:
  - Fine grained, flexible, configurable policies
  - Distinguishes between code of varying degrees of trust
- But…
  - How do we understand what the policy is?
  - Semantics tied to the operational behavior of the program (defined in terms of stacks!)
  - Changing the program (e.g. optimizing it) may change the security policy
  - Policy is distributed throughout the software, and is not apparent from the program interfaces.
  - Is it any good?

  - It's not complete!
Problem with Stack Inspection

```java
main(...){
    fp = new FilePermission("/home/stevez/*", "write,");
    sm.enablePrivilege(fp);
    fileWrite(UntrustedApplet.getFileName(), "xxxxxx");
}
```
Problem with Stack Inspection

main(...){
    fp = new FilePermission("/home/stevez/**", "write,");
    sm.enablePrivilege(fp);
    fileWrite(UntrustedApplet.getFileName(), "xxxxxx");
}
Problem with Stack Inspection

```java
String getFileName() {
    return "/home/stevez/important.txt";
}
```

```java
main(...){
    fp = new FilePermission("/home/stevez/*", "write,...");
    sm.enablePrivilege(fp);
    fileWrite(UntrustedApplet.getFileName(), "xxxxxx");
}
```
Problem with Stack Inspection

```
main(...){
    fp = new FilePermission("/home/stevez/*", "write,...");
    sm.enablePrivilege(fp);
    fileWrite("/home/stevez/important.txt", "xxxxxx");
}
```
Problem with Stack Inspection

```java
void fileWrite("/home/stevez/important.txt", "xxxxxx") {
    fp = new FilePermission(".../important.txt", "write")
    sm.checkPermission(fp);
    /* ... write s to file filename ... */
}

main(...){
    fp = new FilePermission("/home/stevez/*/", "write,...")
    sm.enablePrivilege(fp);
    fileWrite("/home/stevez/important.txt", "xxxxxx");
} 
```
Stack Inspection Research

• A Systematic Approach to Static Access Control
  François Pottier, Christian Skalka, Scott Smith

• Stack Inspection: Theory and Variants
  Cédric Fournet and Andrew D. Gordon

• Understanding Java Stack Inspection
  Dan S. Wallach and Edward W. Felten
  – Formalize Java Stack Inspection using ABLP logic