

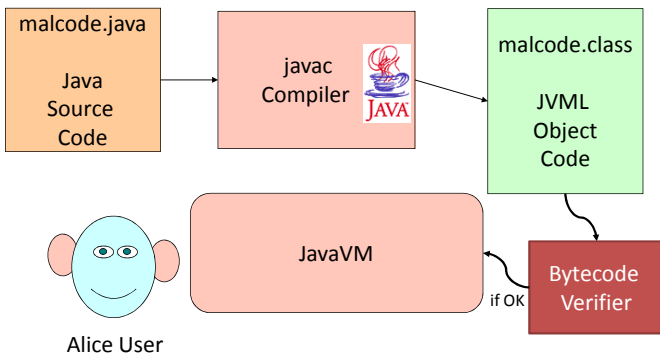
## Plan for Today

- Project Ideas
- Low-level security: “Type Safety”
- Trusted Computing Base (Voting)
- High-level security: Policy

### Project Teams

James Blanton, Sam Herder, Michael Kalish  
 Jeremy Brown, Klaus Dollhopf, Joseph Featherston, Charles Hern, John Marion  
 Joseph Borja, Erik Lopez, Brian Noh, Jonathan DiLorenzo  
 Jiamin Chen, Elisabeth Sparkman, Yixin Sun  
 Michael Dewey-Vogt  
 Hanna Oh  
 Alex Wallace

## Recap: Java Platform



## JVML Instruction Set

pushing constants	20	getstatic, putstatic	2
<b>loads, stores</b>	<b>66</b>	newarray, anewarray, multianewarray, arraylength	<b>4</b>
pop, dup, swap, etc.	9	invoke methods, throw	5
arithmetic	37	new	1
conversion (e.g., i2l)	15	getfield, putfield	2
comparisons (lcmp)	5	checkcast	1
goto, jsr, goto_w, jsr_w, ret	5	instanceof	1
tableswitch, lookupswitch	2	<b>monitorenter, monitorenter</b>	2
returns (e.g., ireturn)	6	wide	1
conditional jumps (ifeq, ifnull, ifnonnull)	16	nop, breakpoint, unused, implementation dependent	5

(205 out of 256 possible opcodes used)

## Why so many loads and stores?

### Instructions are **typed**

aload <index>      load **Object** from variable index  
 iload <index>      load **int** from variable index  
 fload <index>      load **float** from variable index  
 dload <index>      load **double** from variable index

### Minimizing class file size

aload\_0, aload\_1, aload\_2, aload\_3  
 same for other types and stores

### Array loads and stores

Even more types (char, boolean, short)

## Bytecode Verifier

- Checks class file is formatted correctly
- Checks JVM code satisfies safety properties  
**Simulates** program execution to know types are correct, **but doesn't need to examine any instruction more than once**

This is what we win by having static typing!

## Running Mistyped Code

```
> java Simple
Exception in thread "main" java.lang.VerifyError:
(class: Simple, method: main signature:
([Ljava/lang/String;)V)
Register 0 contains wrong type
```

## Verifying Safety Properties

*isdd*

### Type safe

Stack and variable slots must store and load as same type

### Memory safe

Must not attempt to pop more values from stack than are on it  
Doesn't access private fields and methods outside class implementation

### Control flow safe

Jumps must be to valid addresses within function, or call/return

## Wait a sec...

### Hopelessness of Analysis

It is impossible to correctly determine if any interesting property is true for an arbitrary program!

The Halting Problem: it is impossible to write a program that determines if an arbitrary program halts.

from Class 6:

## Making Verification Easier

Class files include lots of extra information to make verification easier

```
public class Simple {
    static public int add (int a, int b) {
        return a + b;
    }
}
```

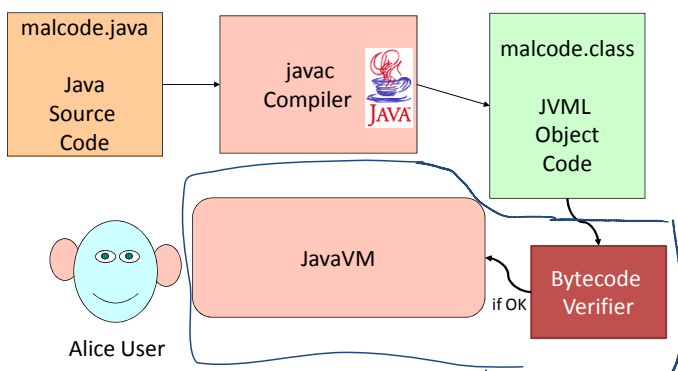
Even with this help there are many "correct" JVMIL programs that would not pass the verifier! (but every program produced by Java compiler should pass)

```
> javac Simple.java
> javap -verbose -c Simple
public class Simple extends java.lang.Object {
    public Simple();
    /* Stack=1, Locals=1, Args_size=1 */
    public static int add(int, int);
    /* Stack=2, Locals=2, Args_size=2 */
}

Method Simple()
  0 aload_0
  1 invokespecial #1 <Method java.lang.Object()>
  4 return

Method int add(int, int)
  0 iload_0
  1 iload_1
  2 iadd
  3 ireturn
```

## Trusted Computing Base



## Project Ideas

### Project Teams

James Blanton, Sam Herder, Michael Kalish  
Jeremy Brown, Klaus Dollhopf, Joseph Featherston,  
Charles Hern, John Marion  
Joseph Borja, Erik Lopez, Brian Noh, Jonathan DiLorenzo  
Jiamin Chen, Elisabeth Sparkman, Yixin Sun  
Michael Dewey-Vogt  
Hanna Oh  
Alex Wallace

## Voting



## What is the Trusted Computing Base for an election?



“We do have people complain and say they don’t get it, I completely understand what they’re saying, but it’s not something I can control.”

Sheri Iachetta,  
Charlottesville general registrar (on 2006 problems with voting machines displaying “James H. ‘Jim’”)

### How do I know my voting equipment is accurate?

Under the Code of Virginia, the State Board of Elections must approve any mechanical or electronic voting system or equipment before it can be used by any locality.

Each system must successfully complete three distinct levels of testing:

1. **Qualification testing** (testing of hardware and software that may be conducted by Independent Testing Authority);
2. **Certification testing** (to ensure it meets all applicable requirements of the Code of Virginia); and,
3. **Acceptance testing** (conducted by the locality to assure it meets their needs and is identical to the certified system).

[www.sbe.virginia.gov/cms/Election\\_Information/Election\\_Procedures/Index.html](http://www.sbe.virginia.gov/cms/Election_Information/Election_Procedures/Index.html)

## “Independent” Testing

- Done by ITAs paid by vendors
- No vulnerability analysis
- No source code analysis

“Program testing can be used to show the presence of bugs, but never to show their absence!”

Edsger W. Dijkstra

## How could we design elections with smaller Trusted Computing Base?



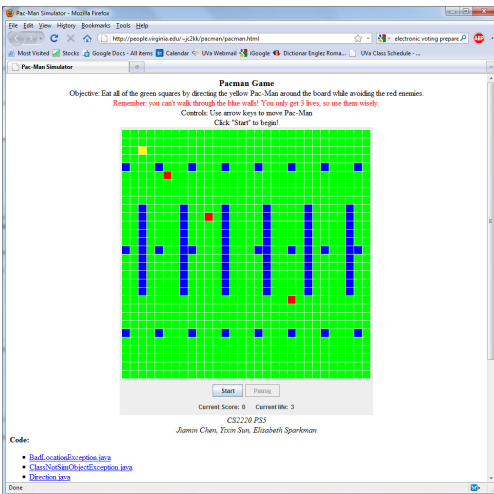
Optical Scan ballots

Can be recounted by humans

## Project Ideas

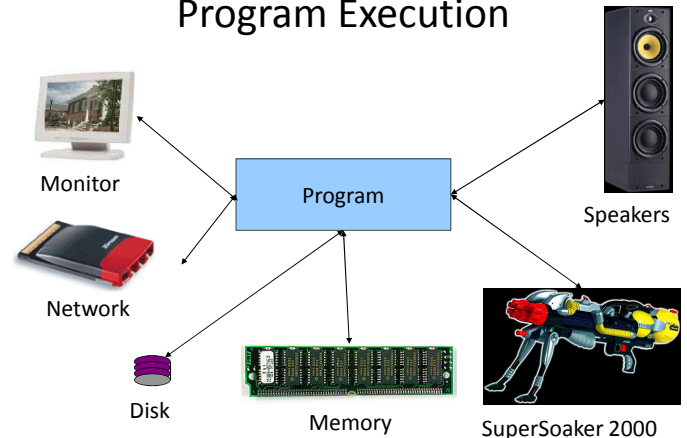
### Project Teams

James Blanton, Sam Herder, Michael Kalish  
Jeremy Brown, Klaus Dollhopf, Joseph Featherston,  
Charles Hern, John Marion  
Joseph Borja, Erik Lopez, Brian Noh, Jonathan DiLorenzo  
Jiamin Chen, Elisabeth Sparkman, Yixin Sun  
Michael Dewey-Vogt  
Hanna Oh  
Alex Wallace

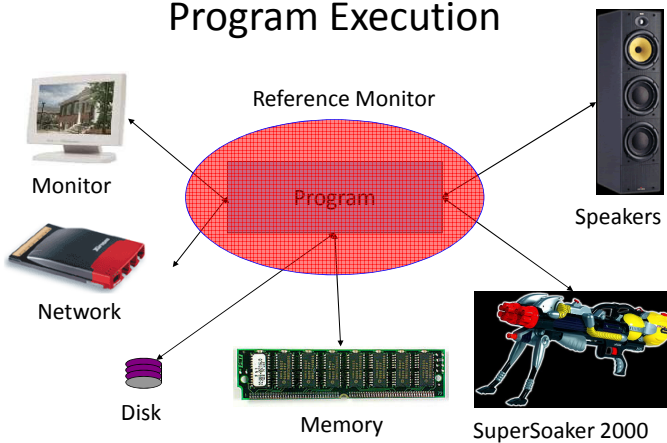


## Java Applet Security

## Program Execution



## Program Execution



## Ideal Reference Monitor

1. Sees <sup>almost</sup> everything a program is about to do before it does it
2. Can instantly and completely stop program execution (or prevent action)
3. Has no other <sup>little</sup> effect on the program or system

Can we build this?

Probably not unless we can build a time machine...

## Ideal Reference Monitor

1. Sees *everything* a program is about to do before it does it
2. Can *instantly* and *completely* stop program execution (or prevent action)
3. Has *no other effect* on the program or system

## Operating Systems

- Provide reference monitors for most security-critical resources
  - When a program opens a file in Unix or Windows, the OS checks that the principal running the program can open that file
- Doesn't allow different policies for different programs
- No flexibility over what is monitored
  - OS decides for everyone
  - Hence, can't monitor inexpensive operations

## Java Security Manager

- (Non-Ideal) Reference monitor
  - Limits how Java executions can manipulate system resources
- User/host application creates a subclass of SecurityManager to define a policy

## JavaVM Policy Enforcement

[JDK 1.0 – JDK 1.1]

*Synchronized*

From java.io.File:

```
public boolean delete() {  
    SecurityManager security =  
        System.getSecurityManager();  
    if (security != null) {  
        security.checkDelete(path);  
    }  
    if (isDirectory()) return rmdir0();  
    else return delete0();  
}
```

**checkDelete** throws a **SecurityException** if the delete would violate the policy (re-thrown by delete)

What could go seriously wrong with this?!

## HotJava's Policy (JDK 1.1.7)

```
public class AppletSecurity  
    extends SecurityManager {  
    ...  
    public synchronized void checkDelete(String file)  
        throws Security Exception {  
        checkWrite(file);  
    }  
}
```

## AppletSecurity.checkWrite

(some exception handling code removed)

```
public synchronized void checkWrite(String file) {  
    if (inApplet()) {  
        if (!initACL) initializeACLs();  
        String realPath = (new File(file)).getCanonicalPath();  
  
        for (int i = writeACL.length ; i-- > 0 ; ) {  
            if (realPath.startsWith(writeACL[i])) return;  
        }  
        throw new AppletSecurityException  
            ("checkwrite", file, realPath);  
    }  
}
```

Note: no checking if not inApplet!  
Very important this does the right thing.

## inApplet

```
boolean inApplet() {  
    return inClassLoader();  
}
```

Inherited from **java.lang.SecurityManager**:

```
protected boolean inClassLoader() {  
    return currentClassLoader() != null;  
}
```

## currentClassLoader

/\*\*

Returns an object describing the most recent class loader executing on the stack.

Returns the class loader of the most recent occurrence on the stack of a method from a class defined using a class loader; returns null if there is no occurrence on the stack of a method from a class defined using a class loader.

\*/

```
protected native ClassLoader currentClassLoader();
```

## Recap

- `java.io.File.delete` calls `SecurityManager.checkDelete` before deleting
- `HotJava` overrides `SecurityManager` with `AppletSecurity` to set policy
- `AppletSecurity.checkDelete` calls `AppletSecurity.checkWrite`
- `AppletSecurity.checkWrite` checks if any method on stack has a `ClassLoader`
- If not, no checks; if it does, checks ACL list

## JDK 1.0 Trust Model

- When JavaVM loads a class from the CLASSPATH, it has no associated ClassLoader (can do anything)
- When JavaVM loads a class from elsewhere (e.g., the web), it has an associated ClassLoader

## JDK Evolution

- JDK 1.1: Signed classes from elsewhere and have no associated ClassLoader
- JDK 1.2:
  - Different classes can have different policies based on ClassLoader
  - Explicit enable/disable/check privileges
  - SecurityManager is now AccessController

## Policy and Mechanism

- AccessController provides a mechanisms for enforcing a security policy
  - Can insert checking code before certain operations are allowed
- A *security policy* determines what the checking code allows

## Android Permissions

The screenshot shows the 'Android Permissions' page on the Android Developers website. The page is titled 'Android Permissions' and has a navigation bar with links for Name, SDK, Dev Guide, Reference, Resources, Videos, and Blog. The main content area is divided into 'Summary' and 'Constants' sections. The 'Constants' section lists various permissions with their descriptions. A red circle highlights the 'BRICK' permission, which is described as 'Required to be able to disable the device (very dangerous)'.

Permission	Description
<code>ACCESS_CHECKIN_PROPERTIES</code>	Allows read/write access to the "properties" table in the checkin database; to change values that get uploaded.
<code>ACCESS_COARSE_LOCATION</code>	Allows an application to access coarse (e.g., Cell-ID, Wi-Fi) location
<code>ACCESS_FINE_LOCATION</code>	Allows an application to access fine (e.g., GPS) location
<code>ACCESS_LOCATION_EXTRA_COMMANDS</code>	Allows an application to access extra location provider commands
<code>ACCESS_MOCK_LOCATION</code>	Allows an application to create mock location providers for testing
<code>ACCESS_NETWORK_STATE</code>	Allows applications to access information about network
<code>ACCESS_SURFACE_FLINGER</code>	Allows an application to use SurfaceFlinger's low level features
<code>ACCESS_WIFI_STATE</code>	Allows applications to access information about Wi-Fi networks
<code>ACCOUNT_MANAGER</code>	Allows applications to call into AccountAuthenticators
<code>AUTHENTICATE_ACCOUNTS</code>	Allows an application to act as an AccountAuthenticator for the AccountManager
<code>BATTERY_STATS</code>	Allows an application to collect battery statistics
<code>BIND_APPWIDGET</code>	Allows an application to tell the AppWidget service which application can access AppWidgets data.
<code>BIND_DEVICE_ADMIN</code>	Must be required by device administration packages to ensure that only the system can interact with it.
<code>BIND_PRINT_SERVICE</code>	Must be required by all CUPS-compatible services to ensure that only the system can bind to it.
<code>BIND_WALLPAPER</code>	Must be required by a WallpaperService class to ensure that only the system can bind to it.
<code>BLUETOOTH</code>	Allows applications to connect to paired Bluetooth devices
<code>BLUETOOTH_ADMIN</code>	Allows applications to discover and pair Bluetooth devices
<code>BRICK</code>	Required to be able to disable the device (very dangerous)
<code>BROADCAST_PACKAGE_REMOVED</code>	Allows an application to broadcast a notification that an application package has been removed.
<code>BROADCAST_SMS</code>	Allows an application to broadcast an SMS receipt notification
<code>BROADCAST_STICKY</code>	Allows an application to broadcast sticky intents.
<code>BROADCAST_WAP_PUSH</code>	Allows an application to broadcast a WAP push receipt notification
<code>CALL_PHONE</code>	Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call
<code>CALL_PRIVILEGED</code>	Allows an application to call any phone number, including emergency numbers, without going through the Dialer user interface

## Charge

Only ask for the Brick permission if you really need it!  
 Only grant the Brick permission to code that you really trust!

"It's better to beg forgiveness than ask permission."  
 Grace Hopper  
 (Applies to most things, but not Java applets.)