CS216: Program and Data Representation
University of Virginia Computer Science
Spring 2006 **David Evans**

Lecture 19: Java Security

PS6 Submission: Only to be eligible for the "Byte Code Wizard" awards. If the web submission is down, you can submit (once) by email.

STEEL PIER
100 years
1898 1998
ATLANTIC PIER AMUSEMENTS, INC.
WELCOME TO THE FAMOUS
STEEL PIER
IN ORDER TO PROTECT OUR GUESTS, PLEASE OBSERVE THE FOLLOWING RULES.
NO ALCOHOLIC BEVERAGES
NO PETS
NO SKATEBOARDS
NO BIKES
NO ROLLER-BLADES
NO RADIOS
NO VULGAR TEE-SHIRTS
NO LOITERING OR SOLICITING
ALL GUESTS ARE REQUIRED TO WEAR SHIRTS AND SHOES.
THANK YOU, STAFF AND MANAGEMENT
MILK SHAKES
TICKETS
SINGLE TICKETS 75¢
35 TICKETS $25
80 TICKETS $50
200 TICKETS $100
ALL DAY RIDE PASS $35 PER PERSON

http://www.cs.virginia.edu/cs216

Running Mistyped Code

```
.method public static main([Ljava/lang/String;)V
   …
   iconst_2
   istore_0
   aload_0
   iconst_2
   iconst_3
   iadd
   …
   return
.end method
```

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

```
> java –noverify Simple
result: 5
```

Running Mistyped Code

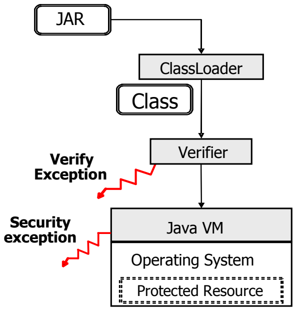
```
.method public static main([Ljava/lang/String;)V
   …
   ldc 216
   istore_0
   aload_0
   iconst_2
   iconst_3
   iadd
   …
.end method
```

```
> java –noverify Simple
Unexpected Signal : EXCEPTION_ACCESS_VIOLATION
(0xc0000005) occurred at PC=0x809DCEB
Function=JVM_FindSignal+0x1105F
Library=C:\j2sdk1.4.2\jre\bin\client\jvm.dll

Current Java thread:
at Simple.main(Simple.java:7)
…

#
# HotSpot Virtual Machine Error : EXCEPTION_ACCESS_VIOLATION
# Error ID : 4F530E43505002EF
# Please report this error at
# http://java.sun.com/cgi-bin/bugreport.cgi
#
# Java VM: Java HotSpot(TM) Client VM (1.4.2-b28 mixed mode)
```

Java Security Architecture

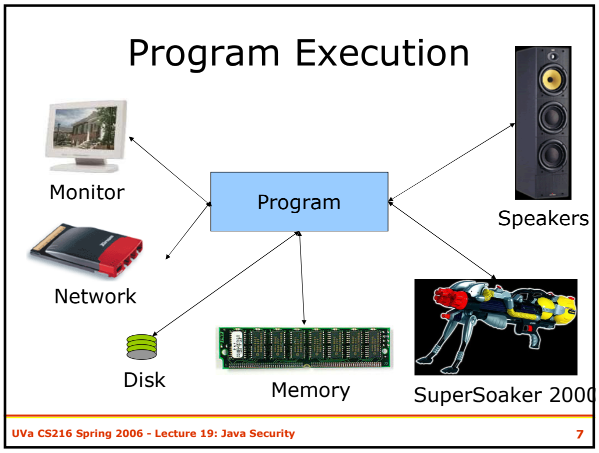


JavaVM

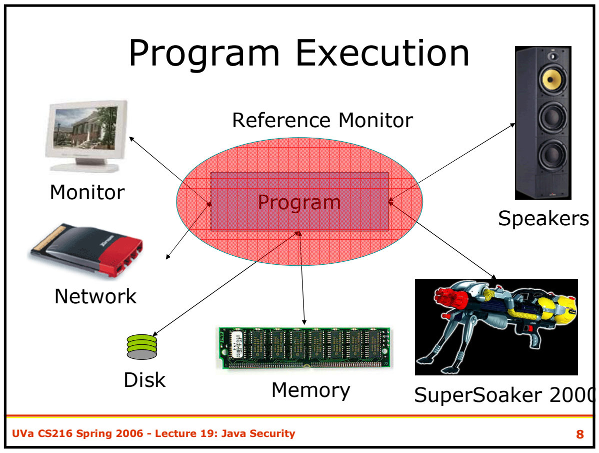
- Interpreter for JVML programs
- Has complete access to host machine: its just a C program running normally
- Bytecode verifier ensures some safety properties, JavaVM must ensure rest:
  - Type safety of run-time casts, array assignments
  - Memory safety: array bounds checking
  - **Resource use policy**

Reference Monitors

Program Execution



Program Execution



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

Can we build this?

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

~~Ideal~~ Real Reference Monitor

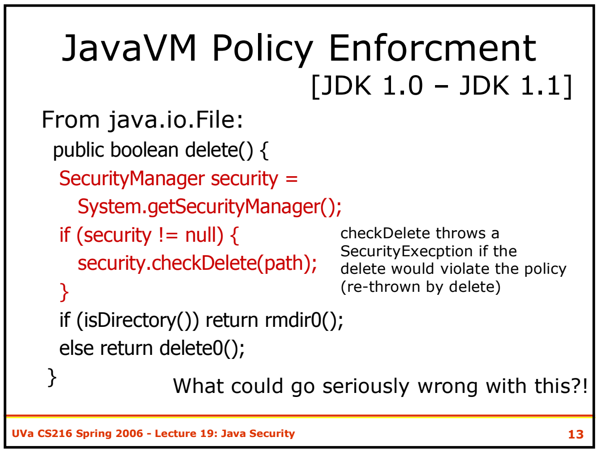
1. Sees ~~everything~~ most things a program is about to do before it does it
2. Can ~~instantly and completely~~ stop ~~program execution~~ (or prevent action)
3. Has ~~no other~~ limited effect on the program or system

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



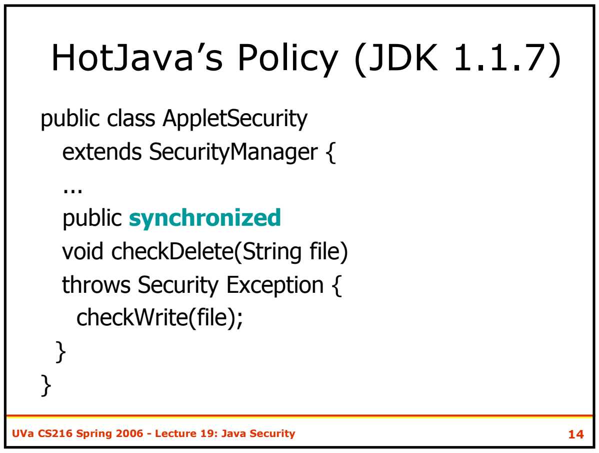
JavaVM Policy Enforcment
[JDK 1.0 – JDK 1.1]

From java.io.File:

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

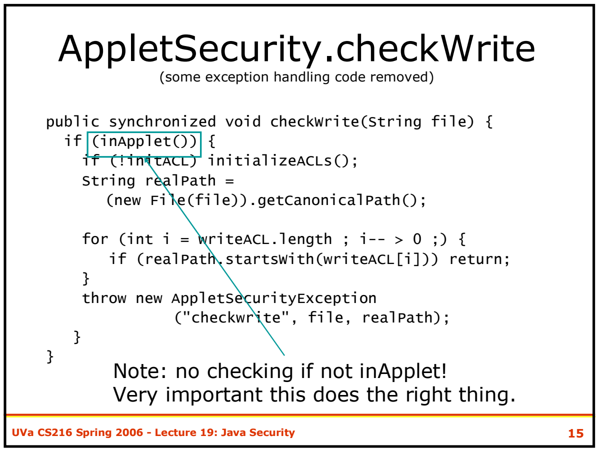
checkDelete throws a SecurityExecption if the delete would violate the policy (re-thrown by delete)

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);
   }
}
```

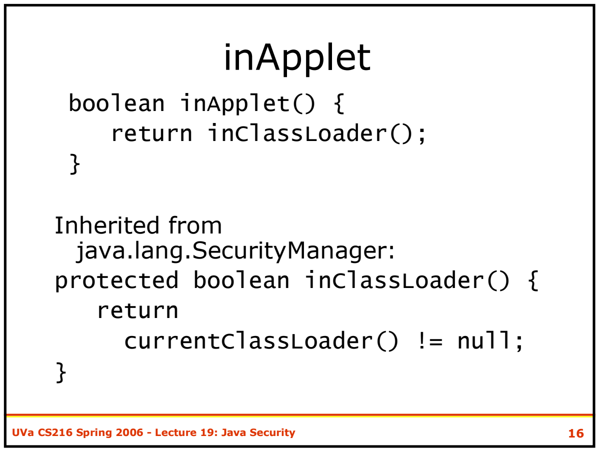


AppletSecurity.checkWrite
(some exception handling code removed)

```
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.

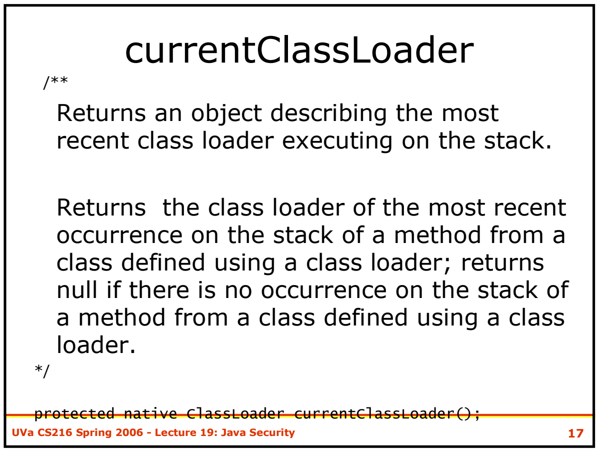


inApplet

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

Inherited from java.lang.SecurityManager:

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



currentClassLoader

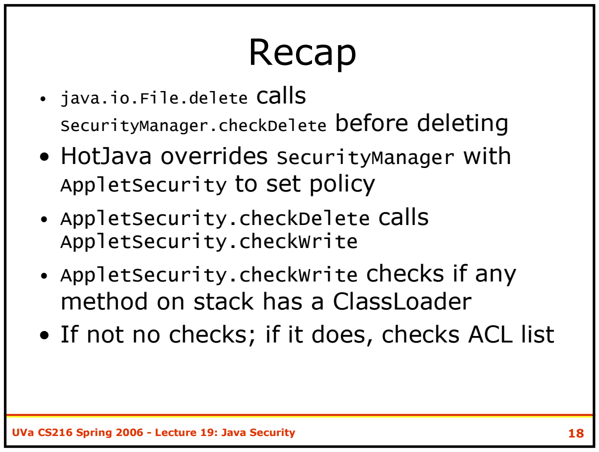
/**

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
  - Explict enable/disable/check privileges
  - SecurityManager is now AccessController

What can go wrong?

- Java API doesn't call right SecurityManager checks (63 calls in java.*)
  - Font loading bug, synchronization
- ClassLoader is tricked into loading external class as internal
- Bug in Bytecode Verifier can be exploited to circumvent SecurityManager
- Policy is too weak (allows damaging behavior)

Example Vulnerability

- Object Creation involves three steps:

  new – create new object reference
  dup – duplicate reference
  invokespecial <> – calls constructor

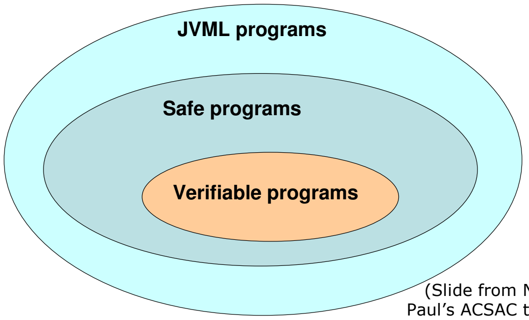
```
new #14 <Class java.lang.StringBuffer>
dup
invokespecial #15 <Method java.lang.StringBuffer()>
```

Object Initialization Vulnerability [lsd-pl.net]

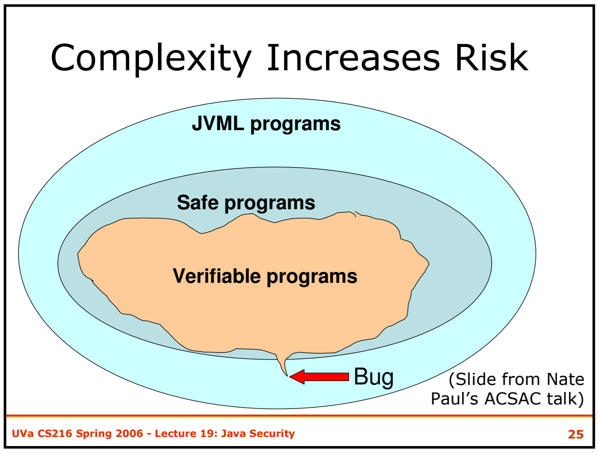
```
class LSDbug extends SecurityClassLoader {
  public LSDbug() {
    try {
      LSDbug(5);
    } catch (SecurityException e) {
      this.loadClass(…);
    }
  }
  public LSDbug (int x) {
    super(); // throws Security Exception
  } }
```

this is used, but not property initialized! Bytecode verifier (old version) didn't make correct checks

Verifier (should be) Conservative

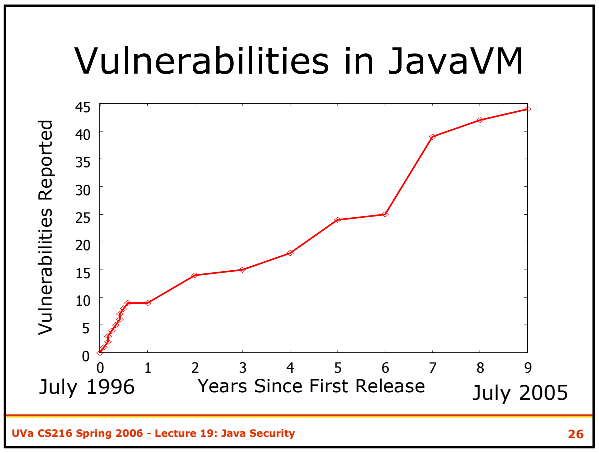


(Slide from Nate Paul's ACSAC talk)

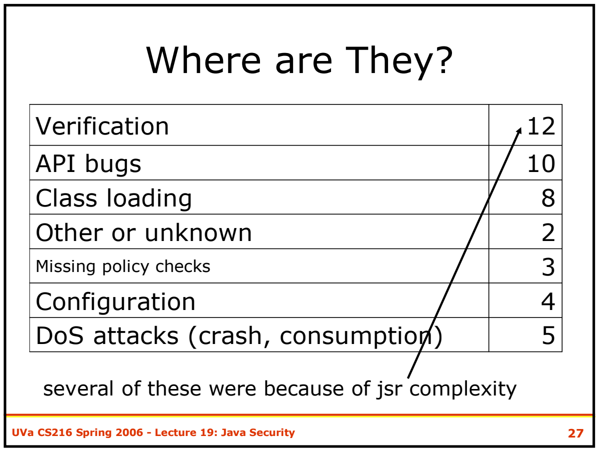


Complexity Increases Risk

(Slide from Nate Paul’s ACSAC talk)



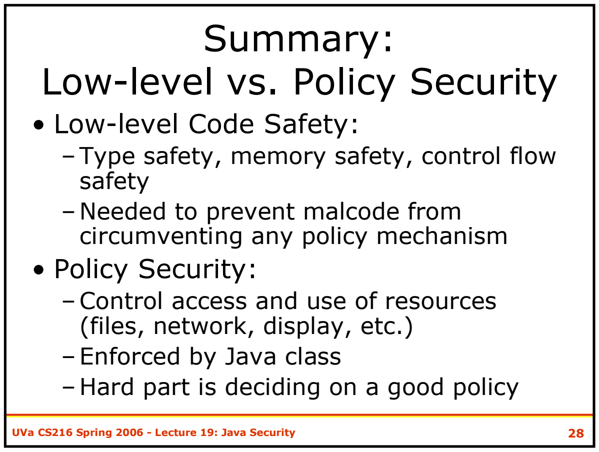
Vulnerabilities in JavaVM



Where are They?

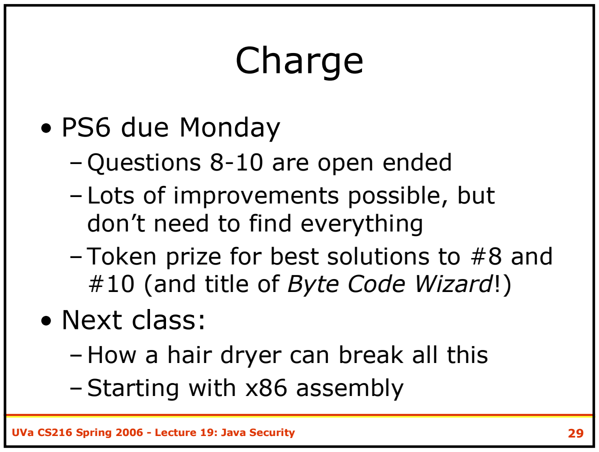
Verification	12
API bugs	10
Class loading	8
Other or unknown	2
Missing policy checks	3
Configuration	4
DoS attacks (crash, consumption)	5

several of these were because of jsr complexity



Summary: Low-level vs. Policy Security

- Low-level Code Safety:
  - Type safety, memory safety, control flow safety
  - Needed to prevent malcode from circumventing any policy mechanism
- Policy Security:
  - Control access and use of resources (files, network, display, etc.)
  - Enforced by Java class
  - Hard part is deciding on a good policy



Charge

- PS6 due Monday
  - Questions 8-10 are open ended
  - Lots of improvements possible, but don’t need to find everything
  - Token prize for best solutions to #8 and #10 (and title of *Byte Code Wizard*!)
- Next class:
  - How a hair dryer can break all this
  - Starting with x86 assembly