

CMSC 433 – Programming Language  
Technologies and Paradigms  
Spring 2004

Security  
April 22, 2004

## So You Want to Download Code

- What if code does bad things?
  - `rm *.*`
- Solution 1: Trust everyone
  - Not such a good idea

2

## So You Want to Download Code

- Solution 2: Trust certain parties
  - E.g., Microsoft, Apple, RedHat
  - Often code will be *signed*
    - Very hard to impersonate trusted party
- Solution 3: Limit your trust
  - Download anyone's code, but...
  - Limit what it can do

3

## Java Byte Code

- javac compiles .java files to .class files
  - .class files contain java byte code
- Suppose you download a .class file
  - javac prevents many kinds of errors, but
  - How do you know .class file came from javac?
    - You don't!
  - Need to re-check that .class files are "type safe"

4

## The Java Verifier

- Input: bytecode
- Output: "pass" or "fail"
  - JVM won't run code that fails the verifier
- Checks for type safety (no seg faults):
  - Methods called with correct #/types of args
  - Methods return values of correct type
  - etc.

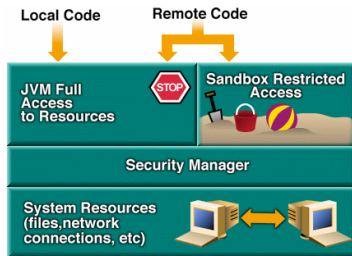
5

## Beyond Type Safety

- Type safety is good, but not enough
  - `rm *.*` doesn't seg fault!
- We need more kinds of restrictions
  - Can't write files
  - Can't connect over the network

6

## JDK 1.0 Security Model



(from java.sun.com 1.2 Security tutorial)

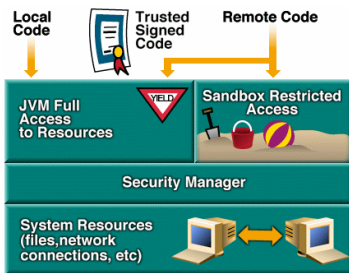
7

## Sandboxing

- Remote code runs in “safe” environment
  - Can’t do much harm
  - Unix example: `chroot`
- Local programs have full access
  - Outside the sandbox

8

## JDK 1.1 Security Model



(from java.sun.com 1.2 Security tutorial)

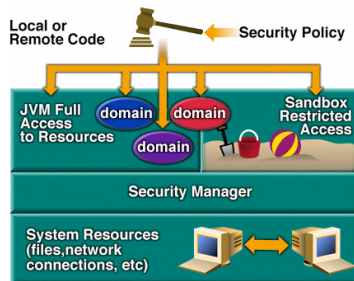
9

## Limitations of 1.0/1.1 Solutions

- 1.0: Remote code can do almost nothing
- 1.1: All or nothing trust
  - One-size fits all solution not good enough
  - Need to support various *security policies*

10

## JDK 1.2 Security Model



(from java.sun.com 1.2 Security tutorial)

11