COMP 150-SEN
Software Engineering Foundations

Java Security

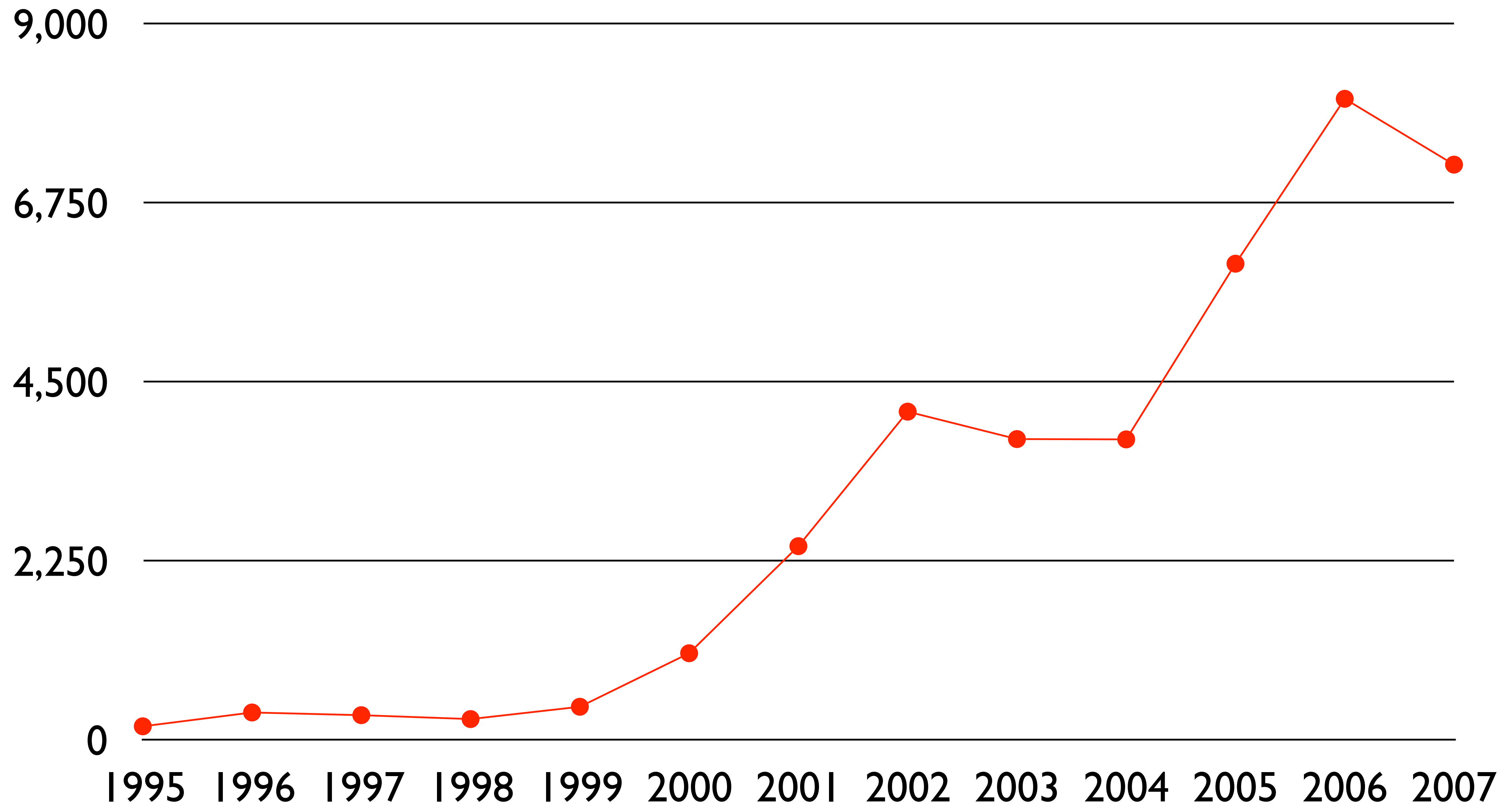
Spring 2019

(Many slides from Steve Zdancewic and Andrew Myers:
http://www.cis.upenn.edu/~cse331/lectures/CSE331-01.pdf
http://www.cs.cornell.edu/andru/pldi06%2Dtutorial/06jun-pldi-tutorial.pdf
http://www.cs.uoregon.edu/research/summerschool/summer04/lectures/myers.pdf)

Motivation

- In the 1970's, computing systems were isolated
  - Software updates infrequent, done by admin
  - Users always trusted code they ran
  - Physical access to machine required
  - Crashes and outages had limited effect
- The Internet has changed all this
  - Depend on software for everyday services
  - Software constantly updated
  - Remote hackers are as close as your neighbor
  - Everything is executable (web pages, email, ...)

CERT Vulnerabilities Reported



What is Security?

- Goal: prevent bad things from happening
  - Clients not paying for services
  - Critical services unavailable
  - Confidential information leaked
  - Important information damaged
  - System used to violate laws
  - Money stolen
  - Loss of value

- Prevent **an adversary** from doing these bad things
  - Normal flaws avoided by users
  - Malicious adversary seeks out weaknesses

When is a Program Secure?

- When it does exactly what it should!
  - But what is it supposed to do?
    - Someone tells us (do we trust them?)
    - We decide ourselves (how do we check?)
    - We write the code ourselves (how much of the software you use have you written?)

- Perfect security does not exist
  - Must trade off performance, cost, usability, functionality
  - When is software “secure enough”?

Example #1: Morris Worm

- 1988: Penetrated estimated 5-10% of 6,000 machines on the internet
- Used a number of clever methods to gain access to a host
  - Brute force password guessing
  - Bug in default sendmail configuration
  - X windows vulnerabilities, rlogin, etc
  - Buffer overrun in fingerd
- Remarks
  - System diversity helped to limit the spread
  - “Root kits” for cracking modern systems are easily available ande largely use the same techniques

Example #2: Love Bug, Melissa

- 1999: Two email-based viruses that exploited
  - A common mail client (MS outlook)
  - Trusting (i.e., uneducated) users
  - VB scripting extensions within messages to
    - Look up address in the contacts database
    - Send a copy of the message to those contacts
- Melissa: Hit an estimated 1.2 million machines
- Love Bug: Caused an estimated $10B in damage
- Remarks:
  - No passwords, crypto, or native code involved

Example #3: Hotmail

- 1999: All Hotmail email accounts fully accessible by anyone, without a password
  - Just change username in an access URL (no programming required!)
- Selected other Hotmail headlines
  - Hotmail bug allows password theft
  - Hotmail bug pops up with JavaScript code
  - Malicious hacker steals Hotmail passwords
  - New security glitch for Hotmail
  - Hotmail bugfix not a cure-all

Example #4: Yorktown

- 1998: "Smart Ship" USS Yorktown suffers propulsion system failure, is towed into Norfolk Naval Base
- Cause: computer operator accidentally types a zero, causing divide-by-zero error that overflows database and crashes all consoles
- Problem fixed **two days** later

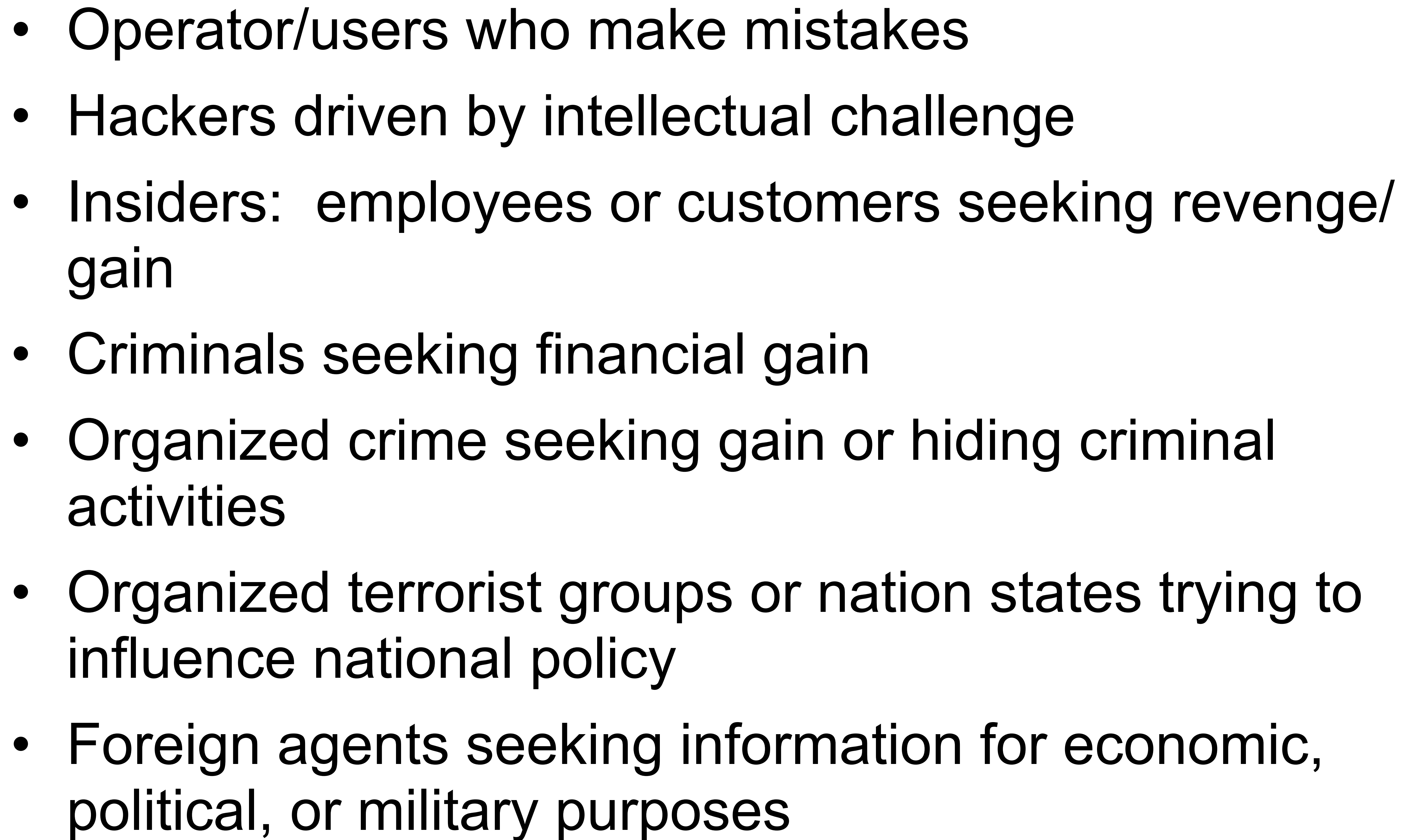
Example #5: Samy Worm

- 2005: MySpace user Samy Kamkar found cross-site scripting vulnerability in MySpace
- Users who visited his page
  - Became friends with him
  - Displayed “but most of all, Samy is my hero” on their profile
  - New user who viewed profile infected
- Within 20 hours, >1,000,000 users infected

Example #6: Insiders

- Average cost of an outsider penetration is $56,000
- Average insider attack cost $2.7 million
  - (~2004, Computer Security Institute/FBI)
- Rogue trader cost Société Générale €3.7B!
  - Most likely cause of stock market sell-off in January
- 63% of companies surveyed reported insider misuse of their organization's computer systems
  - (WarRoom Research)
- Some attacks: Backdoors, "Logic bombs", Hiding data hostage with encryption, Reprogramming cash flows
- Attacks may use legitimately held privileges!
- Many attacks (90%?) go unreported

Who are the Attackers?



- Operator/users who make mistakes
- Hackers driven by intellectual challenge
- Insiders: employees or customers seeking revenge/ gain
- Criminals seeking financial gain
- Organized crime seeking gain or hiding criminal activities
- Organized terrorist groups or nation states trying to influence national policy
- Foreign agents seeking information for economic, political, or military purposes
- Tactical countermeasures intended to disrupt military capability

What are the Vulnerabilities?

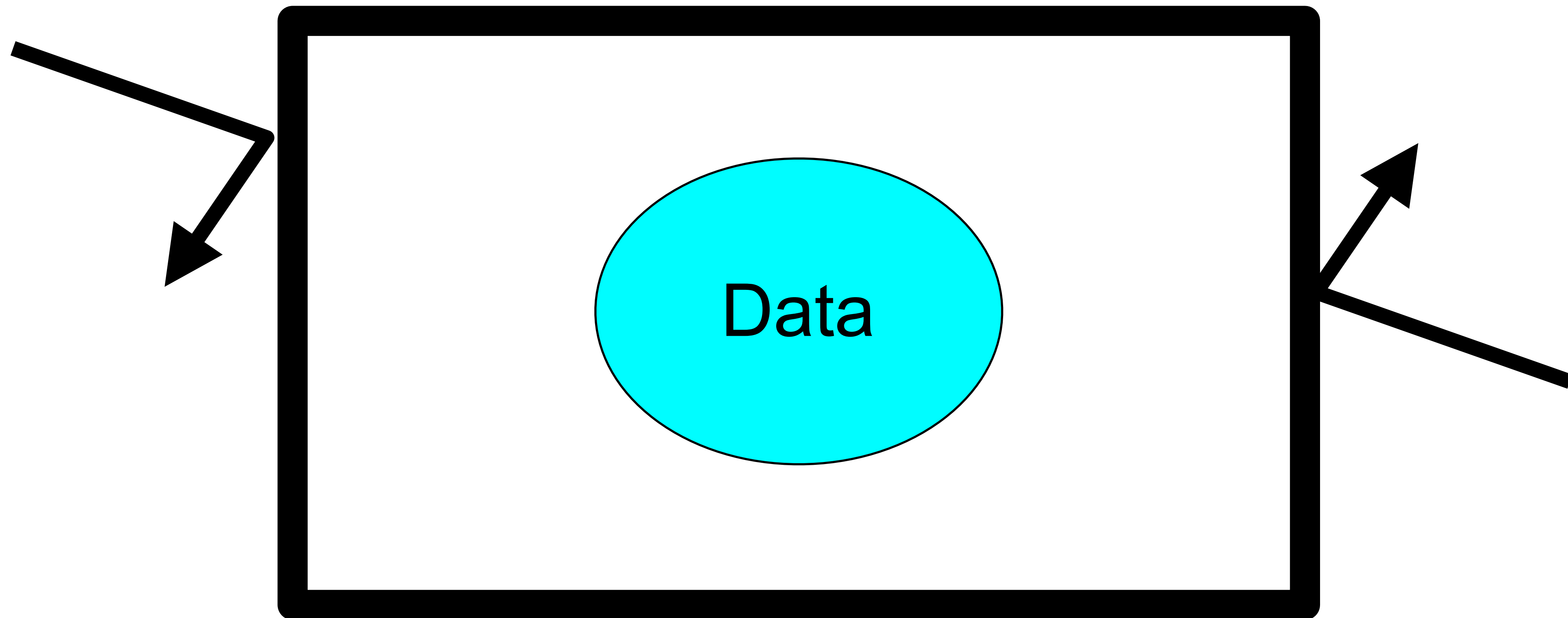
- Poorly-chosen passwords
- Software bugs
- Automatically running active content
  - Macros, scripts, Java programs
- Incorrect configuration
  - File permissions
  - Administrative privileges
- Untrained users/system admins
- Trap doors (intentional security holes)
- Unencrypted communication
- Limited Resources

What are the Attacks?

- Password guessers/crackers
- Viruses
- Worms
  - Viruses require user interaction, worms don't
- Trojan Horses
- Root kits
- Phishing
- Packet sniffers
- Denial of service

Integrity

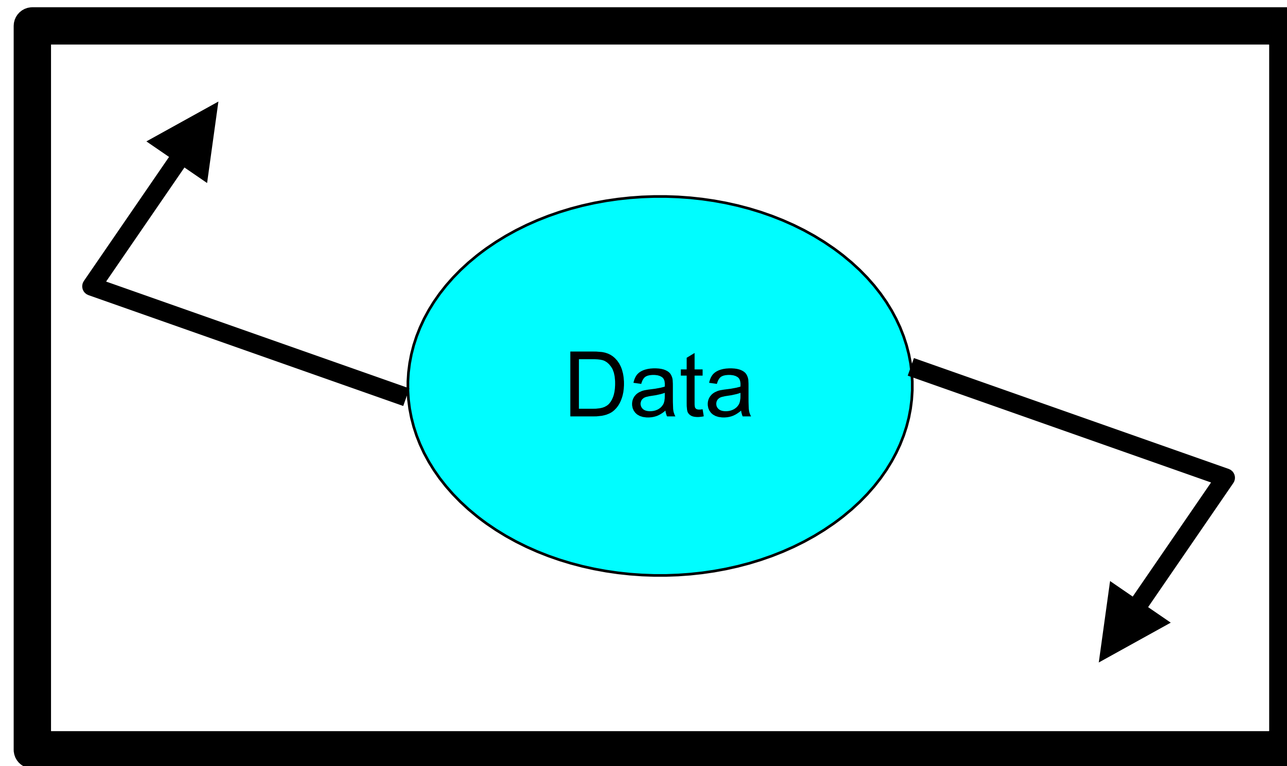
- No improper modification of data



  - E.g., account balance updated only by authorized transactions, only you can change your password
  - Integrity of security mechanism is crucial
  - Enforcement: access control, digital signatures, ...

Confidentiality

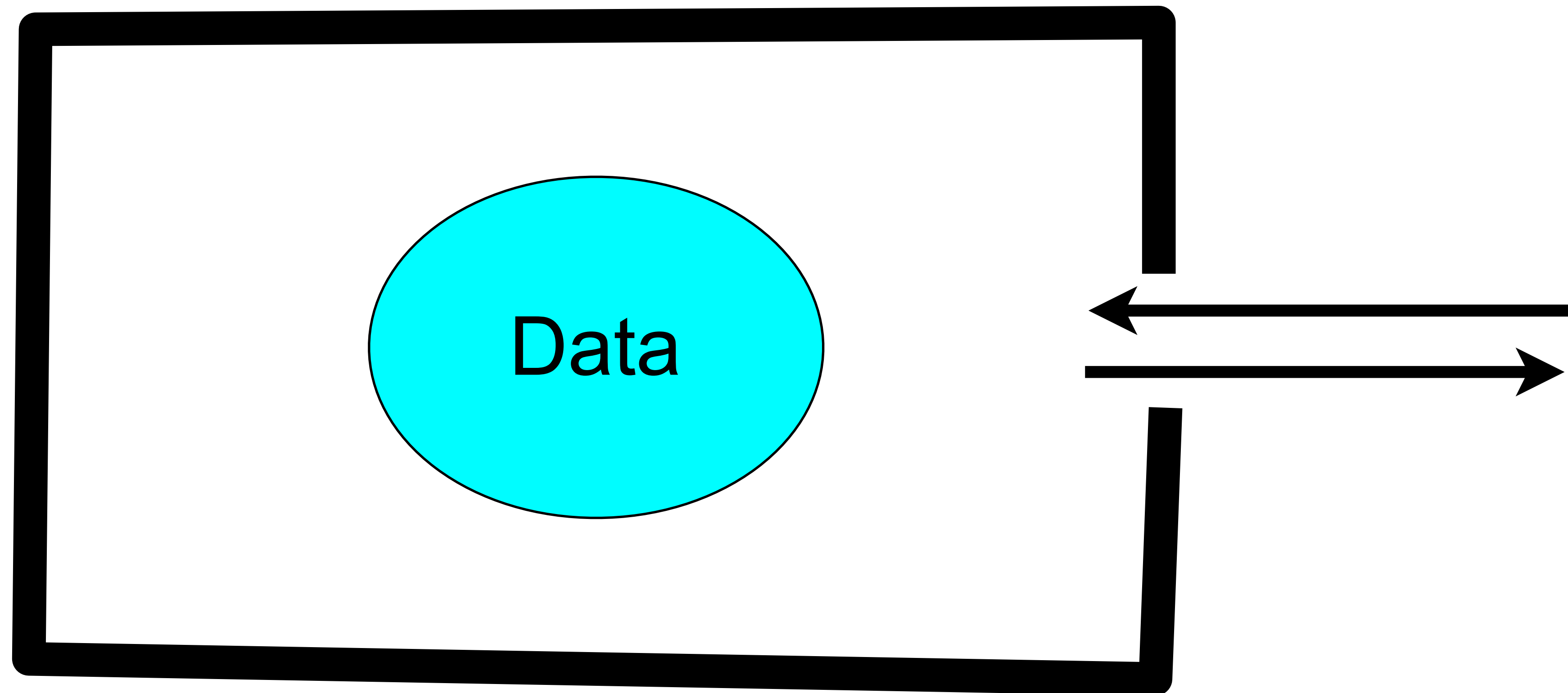
- Protect information from improper release



  - Limit knowledge of data or actions (secrecy)
    - E.g., D-Day attack date, contract bids
  - Enforcement: access control, encryption, ...
  - Hard to enforce after the fact...

Availability

- System must respond to requests



  - I.e., do not ensure confidentiality and integrity by unplugging your computer!

Other Properties

- Privacy: prevent misuse of personal information
- Anonymity: prevent connection from being made between identity of actor and actions

- Attack detection
  - Monitor, intrusion detection
- Recovery from attacks
  - Traceability and auditing of security-relevant actions

- Others?

Project Goals Conflict with Security

- Functionality
- Usability
- Efficiency
- Time-to-market
  - Hard to achieve these and be secure
- Risk assessment
  - *What* needs to be protected, from *whom*, and for *how long*? What is the protection *worth*?

The Java Security Problem

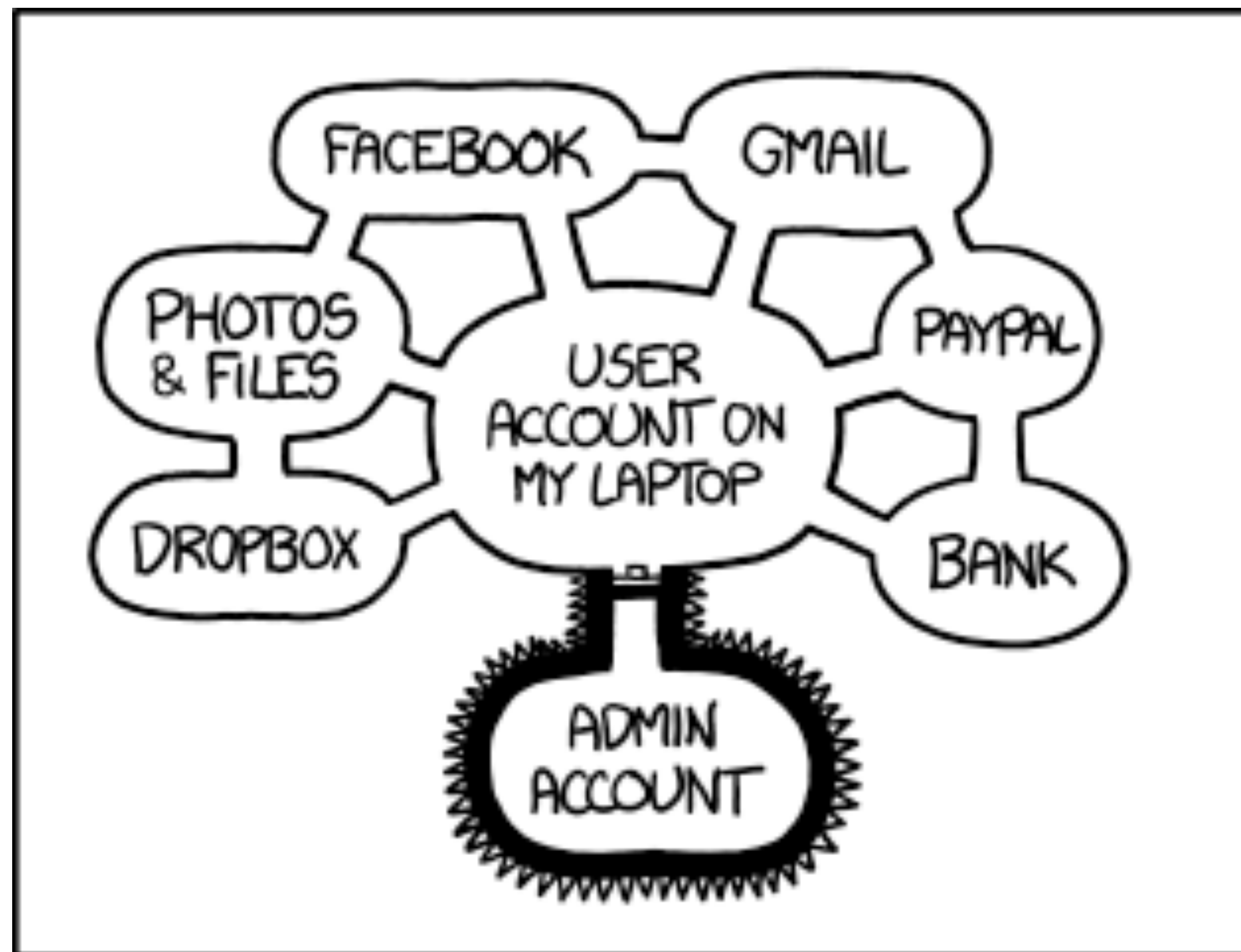
- Java originally envisioned as a mobile code platform
  - Visit a web page, download Java code, run it
  - What if the code does bad things?
    - rm *.*
  - Solution 1: Trust everyone
    - Not such a good idea

The Java Security Problem (cont'd)

- Solution 2: Trust certain parties
  - 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
  - E.g., can't write data over the network
  - E.g., can't write files to disk

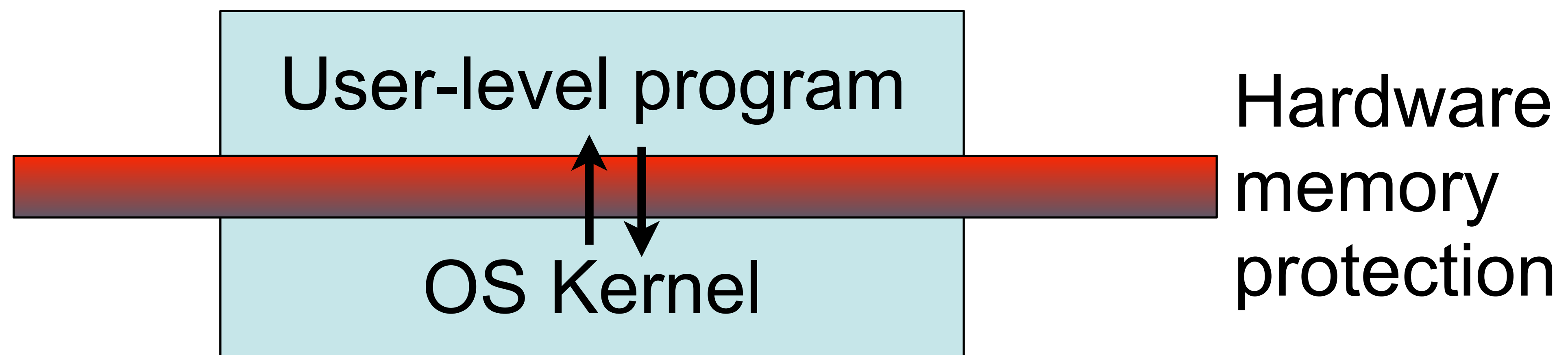
Application Security



https://xkcd.com/1200/

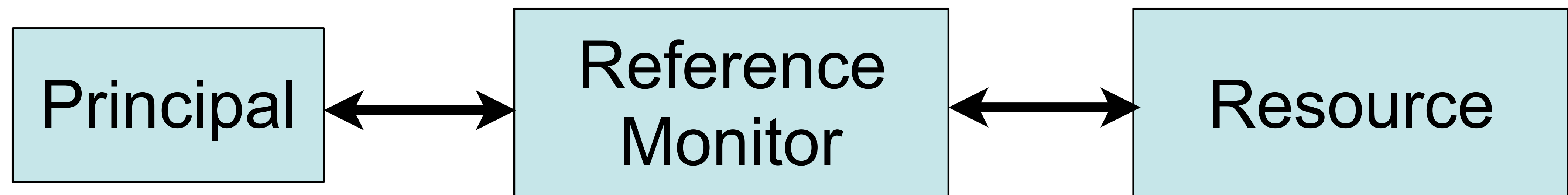
Conventional OS Security

- Model: Program is black box
- Program talks to OS via system calls
  - Protected interface
  - Multiplexes hardware
  - Isolates processes from each other
  - Restricts access to resources (sockets, files, etc)
- Language-independent, simple, limited

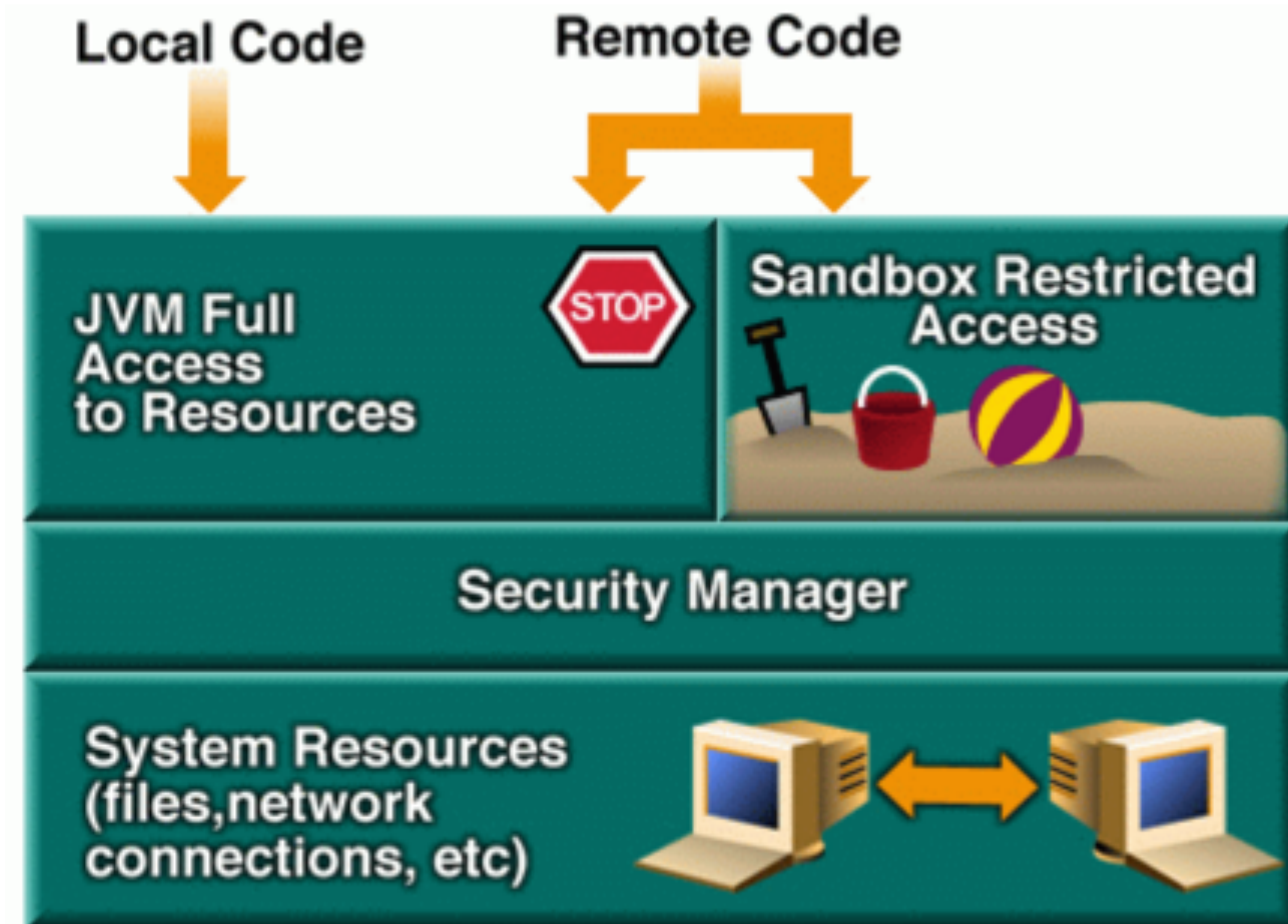


Access Control Model

- The standard way to prevent “bad things”
- *Principals* make *requests* to access resources (*objects*)
- *Reference monitor* (e.g., kernel) permits or denies request



JDK 1.0 Security Model



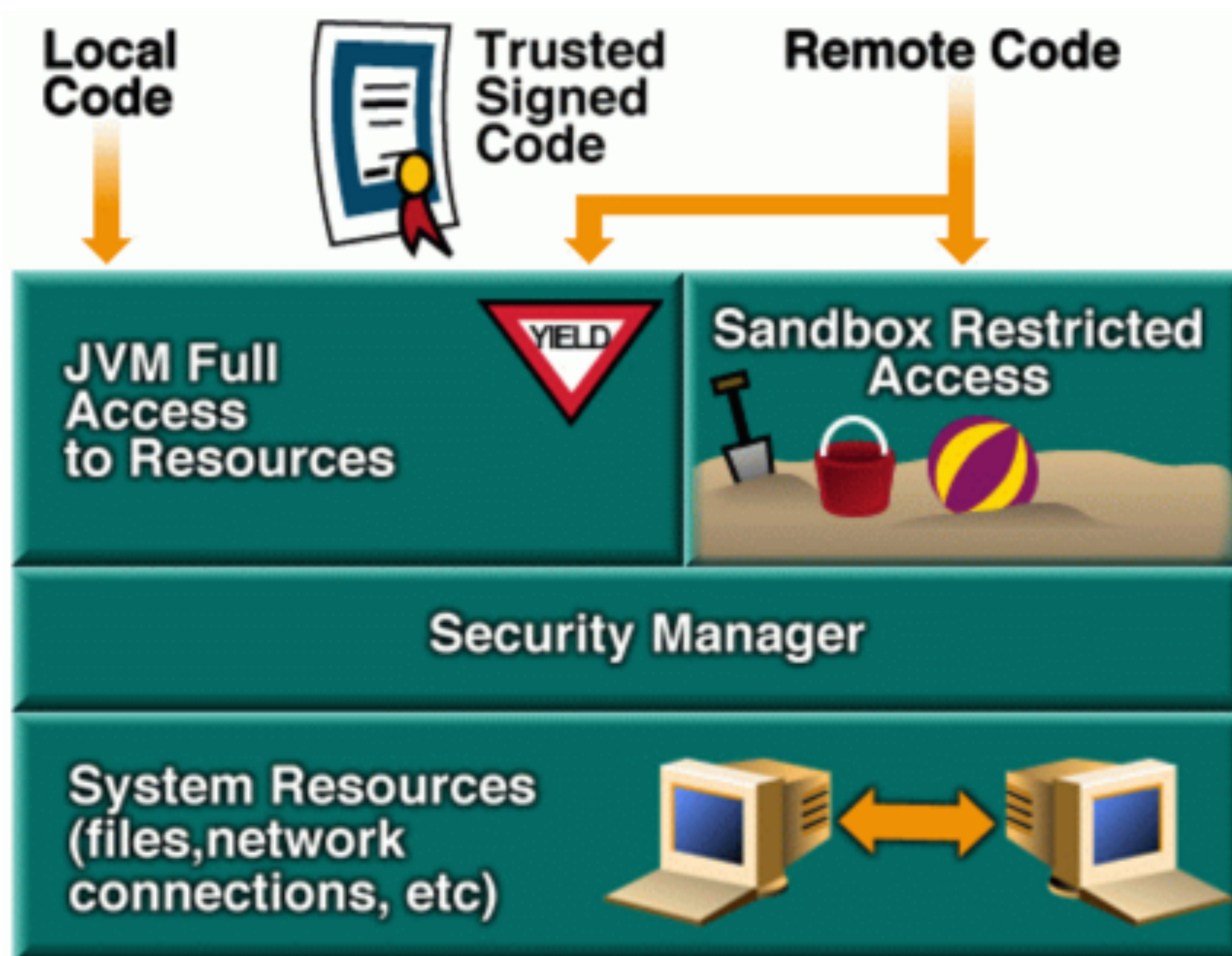
(from java.sun.com 1.2 Security tutorial)

Sandboxing

- Remove code runs in “safe” environment
  - Can’t do much harm
  - Unix: “chroot” command

- Local programs have full access
  - Outside the sandbox

JDK 1.2 Security Model

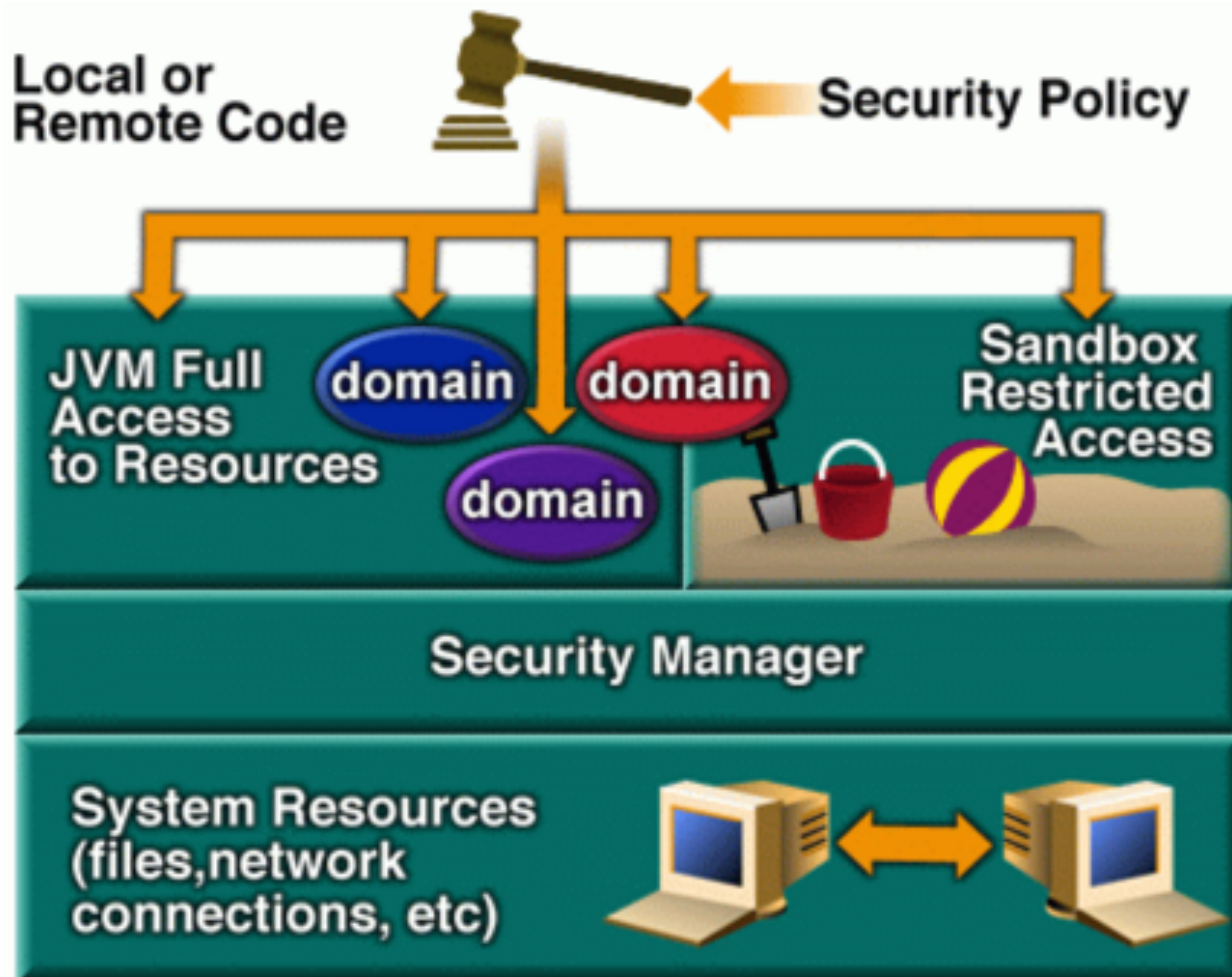


(from java.sun.com 1.2 Security tutorial)

Limitations

- 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*

JDK 1.2 Security Model



(from java.sun.com 1.2 Security tutorial)

Policy File

- Describes what principals are allowed to access which resources

```
grant  {
  permission java.util.PropertyPermission "port", "read";
};
```

- Grants every program the ability to read `“port”` passed as a `-Dport=...` option to Java

Security Managers

- To enable security, install a security manager
  - Without this, no security checks performed
  - There is *always* a security manager installed when running an applet in a web browser

```
System.setProperty("java.security.policy", "java.policy");
  if (System.getSecurityManager() == null)
       System.setSecurityManager(new SecurityManager())
```

- Now we can only invoke ops based on policy

```
int port =
  Integer.getInteger("port", 1099).intValue();  /* OK */
int foo =
  Integer.getInteger("foo", 3).intValue();  /* ERROR */
```

Policy Files Can Distinguish

- Gives `perm1` to methods digitally signed by `K1`
- Gives `perm2` to methods from class files on the local file system
- Gives `perm3` to methods downloaded from `www.evil.com`

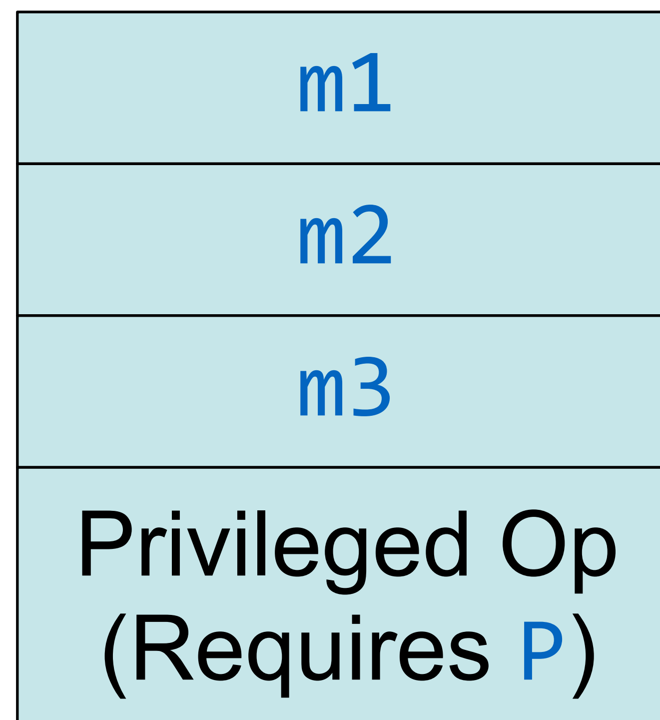
```
grant signedBy K1 {
  permission perm1;
};
grant codeBase “file:/-” {
  permission perm2;
};
grant codeBase “http://www.evil.com” {
  permission perm3;
};
```

The Confused Deputy Problem

- Consider the following situation
  - Method `m1` does not have permission `P`
  - Method `m2` has permission `P`
  - Method `m1` calls `m2`
    - Thus, method `m2` may inadvertently perform operations requiring privilege `P` on `m1`'s behalf
- This is called the *confused deputy problem*
- Java "solves" this problem using stack inspection
  - Which doesn't fully solve the problem
  - And, it's not clear how much of a problem this is...

Stack Inspection

- When Java is about to do a privileged operation, walks back up the call stack and makes sure the current method and *every* caller has the needed permission



Only occurs if m1, m2, and m3 have P

- Why does this not solve "confused deputy" problem?

CheckPermission

- Stack inspection is triggered by calling `checkPermission`

```
PropertyPermission p =
  new PropertyPermission("bar", "read");
AccessController.checkPermission(p);
```

  - Above code checks if current method and all callers have permission to "`read`" property "`bar`"

Privilege Modification

- Sometimes, stack inspection is too restrictive
  - Why?

- In these case, we can call `doPrivileged()` to locally elevate privileges
  - Stack inspections stops at frame that calls `doPrivileged()`
  - Result: stack inspection ignores callers

DoPrivileged Example

```
AccessController.doPrivileged(new Foo());

class Foo implemented PrivilegedAction<T> {
  T run() { .../* do privileged action */... }
}
```

Permissions are Java Objects

- Above code makes a new permission type
  - `p1.implies(p2)` holds if holding `p1` grants `p2`
- Grant block in policy file effectively creates permission object
  - Java uses implies at `checkPermission`

```
class MyPermission extends BasicPermission {
  String name;
  public MyPermission(String name) {
   this.name = name;
  }
  public boolean implies(Permission p2) {
    return name.contains(p2.name);
} }
grant  {
  permission MyPermission “foo”;
};
```

Open Questions

- How does Java stack inspection help with
  - Confidentiality?
  - Integrity?
  - Availability?
- How do you know if you're using stack inspection correctly?
- What other security properties do Java applications want?