

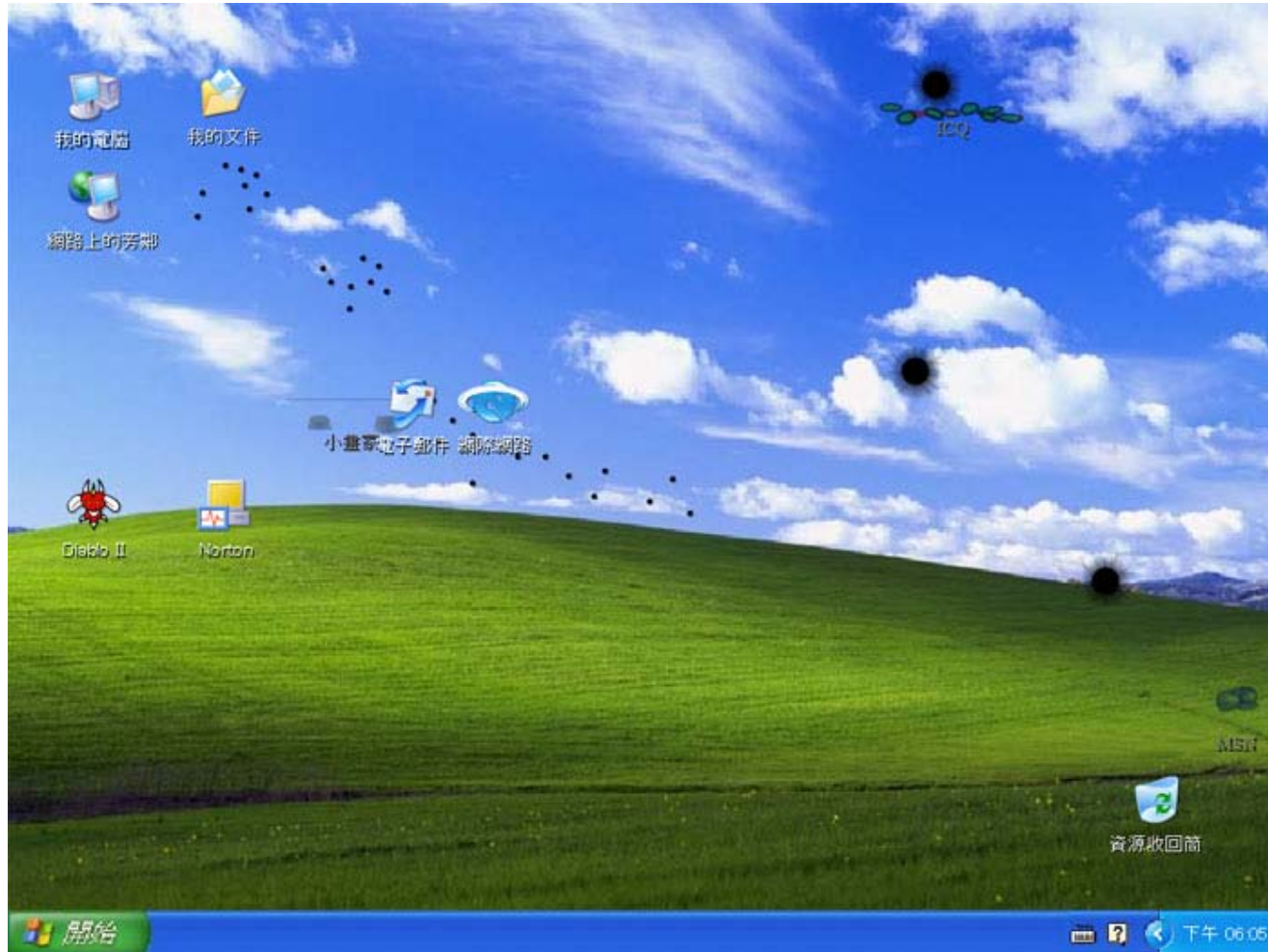
Confinement: The Java Sandbox, Virtual Machines

Lujo Bauer

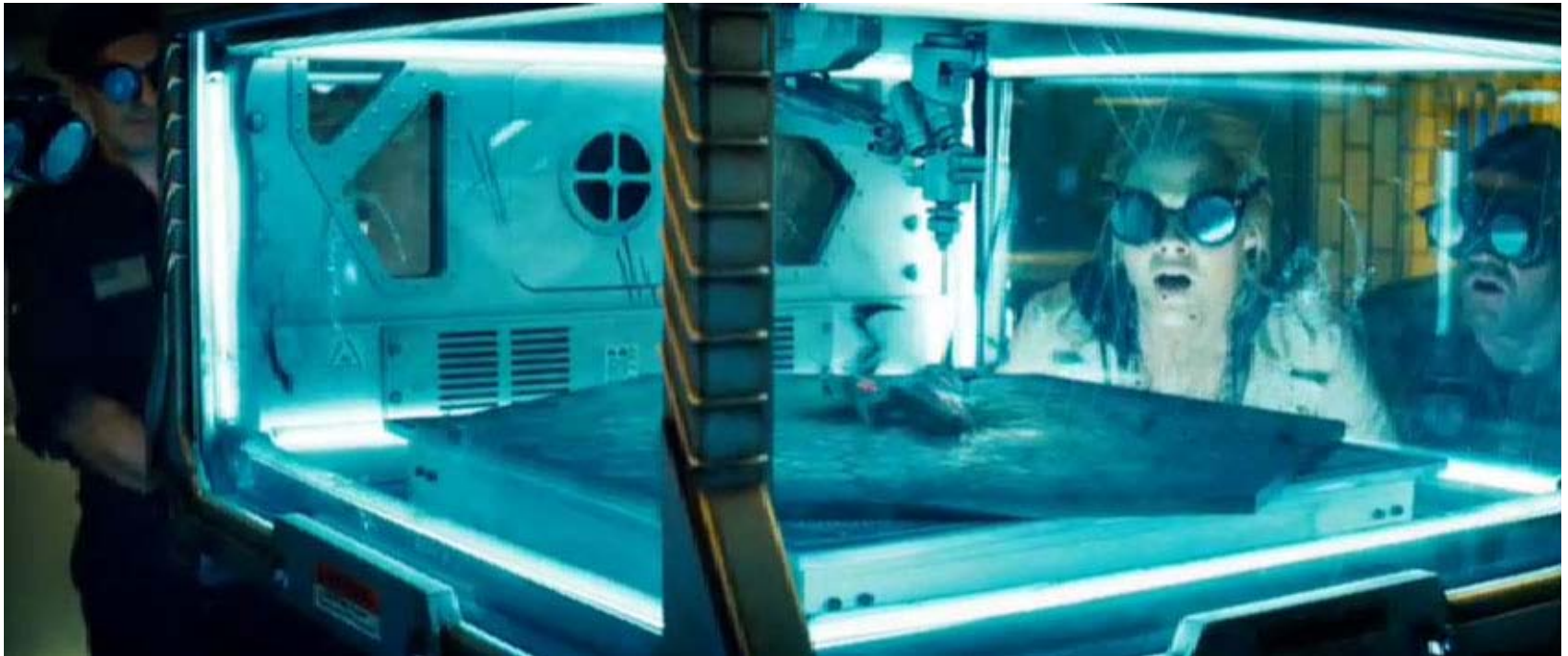
18-732

Fall 2010

When There Is No Protection...



Separation



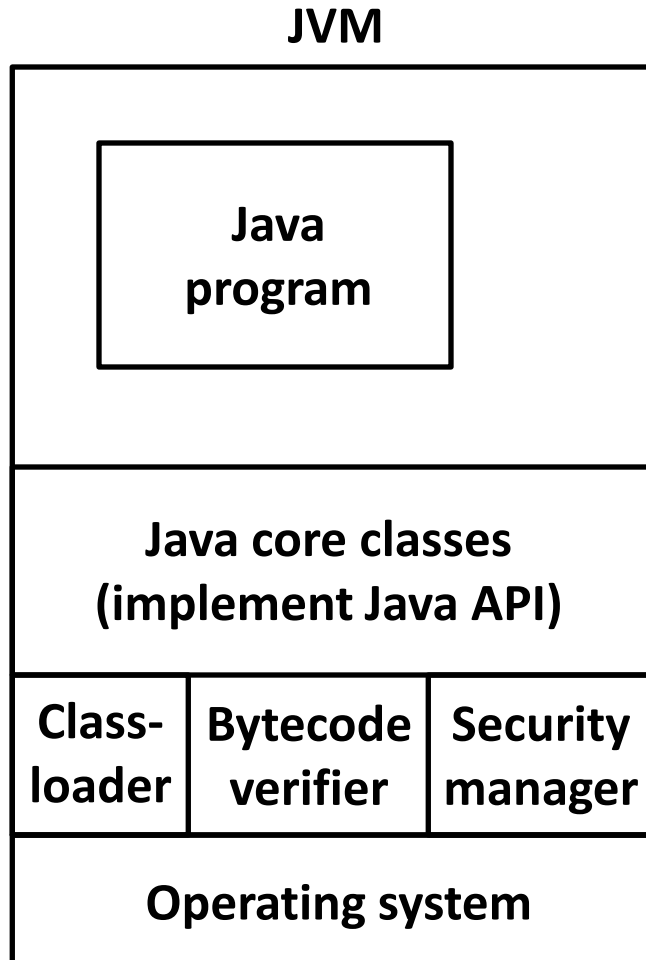
Separation Design Space

- In what dimension should objects be separated?
 - Physical, temporal, logical, cryptographic
- To what degree should objects be kept separate?
 - Complete isolation, ..., limited sharing, usage control, full sharing
- What are the objects that will be kept apart?
 - Parts of programs, programs, sets of programs, OS environments
- What mechanisms will provide separation?
 - Memory protection, sandboxes, virtual machines, ...

Separation Design Space

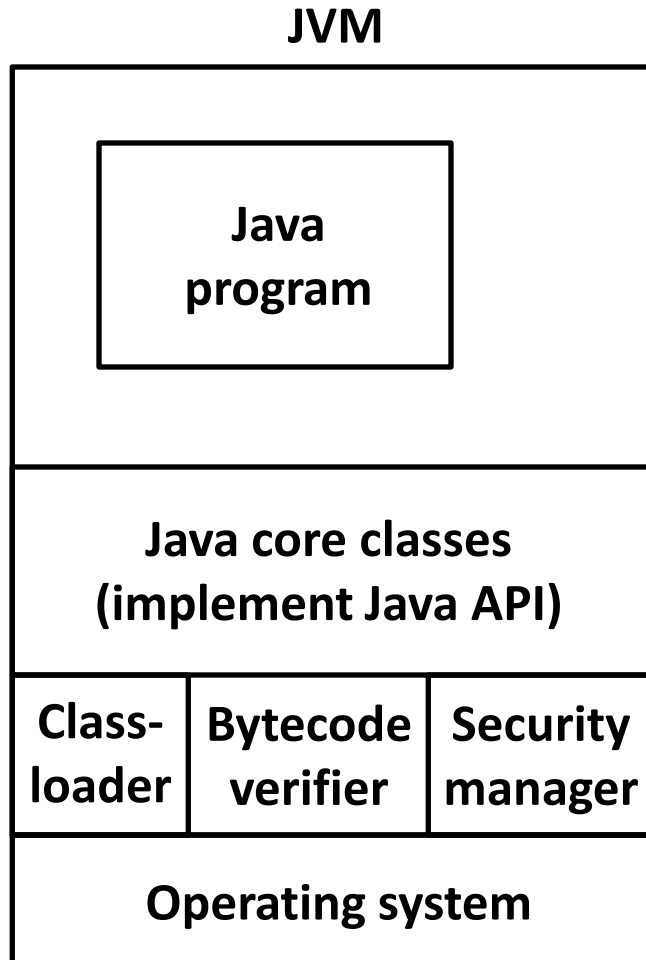
- In what dimension should objects be separated?
 - Physical, temporal, **logical**, cryptographic
- To what degree should objects be kept separate?
 - **Complete isolation, ..., limited sharing, usage control, full sharing**
- What are the objects that will be kept apart?
 - Parts of programs, **programs, sets of programs, OS environments**
- What mechanisms will provide separation?
 - Memory protection, **sandboxes, virtual machines**, ...
- **Today: sandboxing & virtual machines**
 - **Focus on crossing the confinement boundary**

Sandbox #1: The Java Virtual Machine (JVM)



- Java programs use the Java API
 - Can't run on or directly interact with the OS
- Java core classes interface between program and OS/hardware

The Java Virtual Machine (JVM)

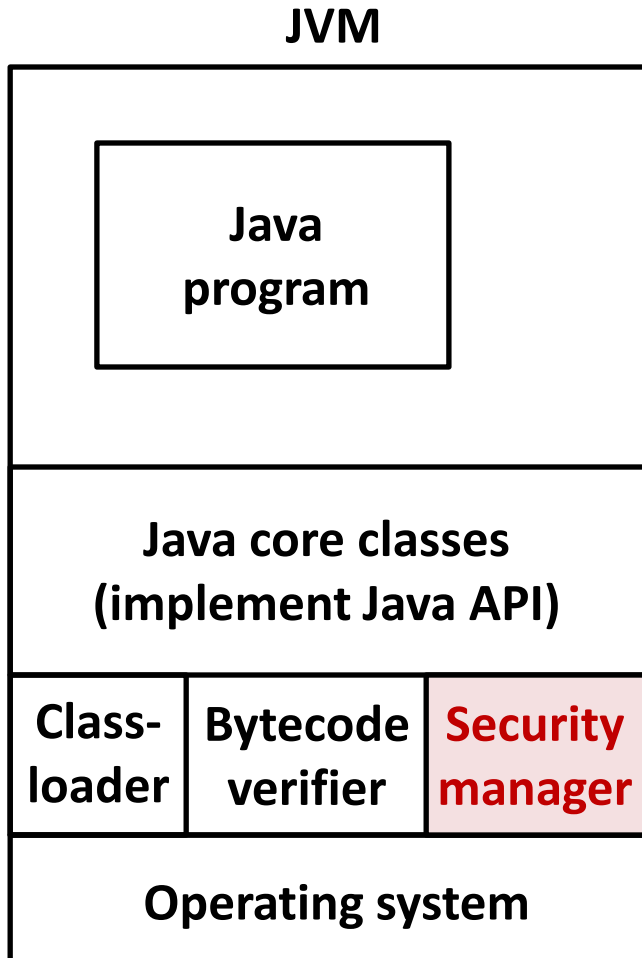


- Security manager mediates programs' attempts to reach outside the sandbox
- Classloader and bytecode verifier ensure that programs don't:
 - Corrupt each other
 - Corrupt the security manager
- Corrupt = interact with in ways not allowed by the API

Type Safety

- Java programs are composed of classes
 - A *class* defines a collection of data fields and functions (methods) that operate on those fields
 - Instances of classes are *objects*
- Type safety
 - Only objects for which the program has a reference can be accessed
 - References cannot be forged
 - Memory cannot be accessed directly
 - A program can perform an operation on an object only if that operation is valid for that object
 - E.g., can't dereference an integer, can't read beyond end of array
- **Type safety is the cornerstone of Java security**

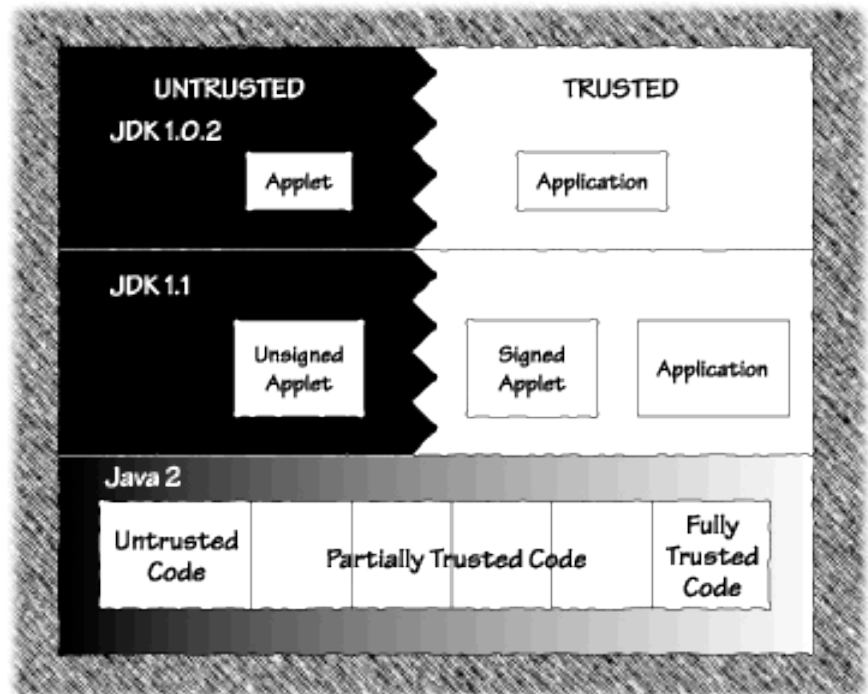
Security Manager



- Security manager essentially is a reference monitor
 - Decides if an applet's requested operation should be allowed
- When an applet makes a potentially dangerous request
 - Java API code invokes the Security Manager
 - Security Manager throws a Security Exception if operation is denied
 - If operation is permitted, the Security Manager returns without exception
 - API code performs requested operation

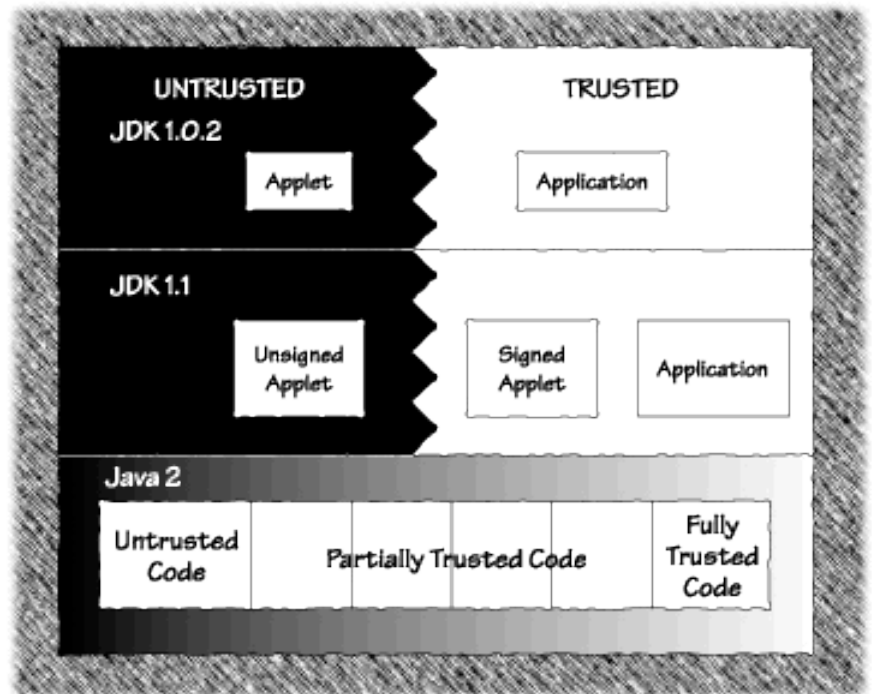
Security Manager Trust Model

- Trust model has become increasingly refined over time
 - JDK 1.0.2
 - Untrusted = any applet
 - Trusted = application
 - JDK 1.1
 - Untrusted = unsigned applet
 - Trusted = signed applet or application
 - JDK 1.2 and beyond
 - Many “shades of gray” in between



Security Manager Trust Model

- Trust model has become increasingly refined over time
- JDK 1.0.2
 - Untrusted = any applet
 - Trusted = application
- JDK 1.1
 - Untrusted = unsigned applet
 - Trusted = signed applet or application
- JDK 1.2 and beyond
 - Many “shades of gray” in between



Security Manager Trust Model

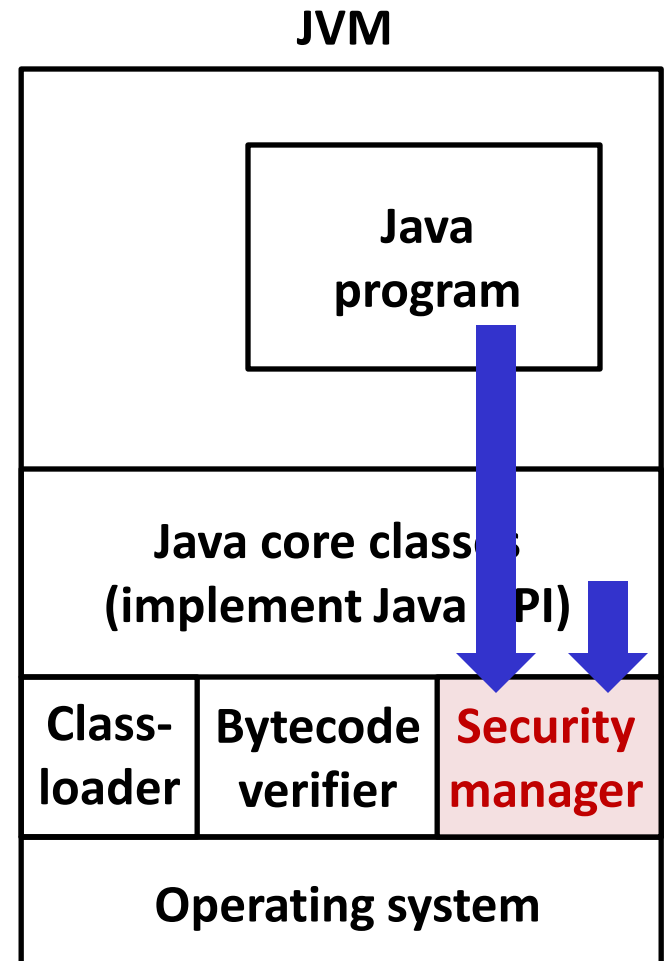
- What untrusted code can do
 - Access CPU and memory to build its objects and execute
 - Connect to the web server from which the applet was downloaded
- What untrusted code can't do
 - Operations on the client's file system such as read, write, delete or rename a file
 - Connect to destinations other than its origin
 - Making critical system calls, such as `System.exit()`
 - Creating a classloader or security manager
 - Other dangerous actions, e.g., manipulate other threads

Security Manager Duties

1. Prevent installation of new class loaders
2. Protect threads and thread groups from each other
3. Control the execution of other application programs
4. Control the ability to shut down the VM
5. Control access to other application processes
6. Control access to system resources such as print queues, clipboards, event queues, system properties, and windows
7. Control file system operations such as read, write, and delete
8. Control network socket operations such as connect and accept

Stack Inspection

- When security manager evaluates a request, what code does it hold responsible for the request?
- Consider a request to open file
 - There are possibly many layers of “library” code between the untrusted applet and the point at which the Security Manager is invoked
 - Which of these calls should be granted?



Simplified Stack Inspection

- Assume two principals (“system” & “untrusted”) and one privilege
- Each Java thread has a run-time stack that tracks method calls
- Every stack frame is labeled with principal and a privilege tag
 - A system class can set the tag while untrusted code cannot
 - Whenever a frame completes its work, the tag disappears
- Per access request
 - Algorithm searches frames from newest to oldest
 - If a frame with “full” tag is first found, then access is permitted
 - If an untrusted frame is encountered, access is denied

Stack Inspection Operations

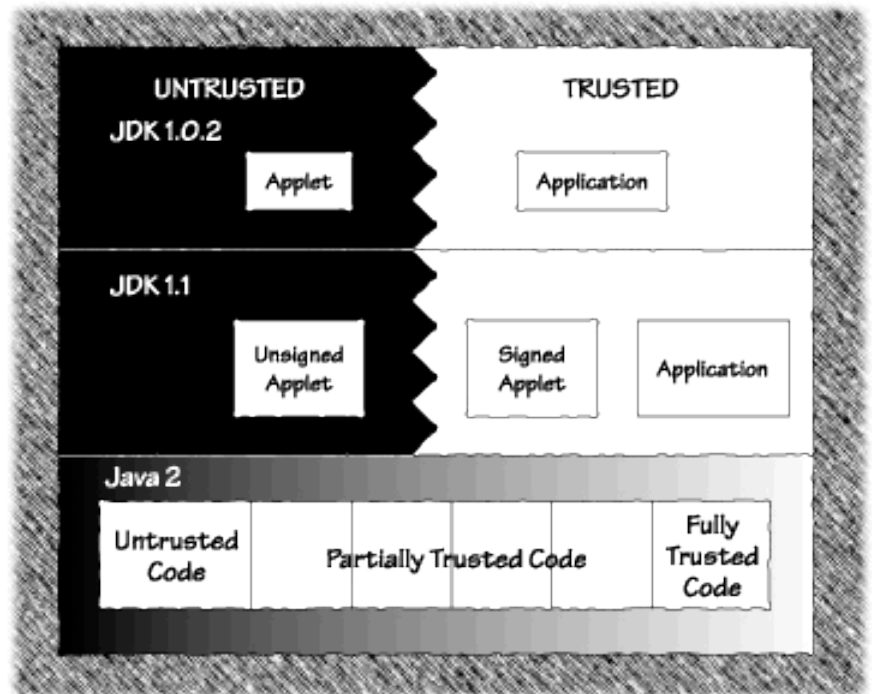
- **enablePrivilege()**
 - When code wants to use a resource R , it must first call **enablePrivilege(R)**
 - If permitted, the current stack frame is annotated with an enabled privilege mark
- **revertPrivilege()**
 - Removes annotation from stack frame
- **disablePrivilege()**
 - Creates a stack annotation to hide the earlier enabled privilege
- **checkPrivilege()**
 - Search stack from the newest to the oldest to check if a frame has proper privilege to make a specific system call

Stack Inspection Algorithm

```
checkPrivilege (target) {  
    // loop, newest to oldest stack frame  
    foreach stackFrame {  
        if (local policy forbids access to target by class  
            executing in stackFrame)  
            throw ForbiddenException;  
        if (stackFrame has enabled privilege for target)  
            return; // allow access  
        if (stackFrame has disabled privilege for target)  
            throw ForbiddenException;  
    }  
    // if we reached here, we fell off the end of the stack  
    if (Netscape 4.0)  
        throw ForbiddenException;  
    if (Microsoft IE 4.0 || Sun JDK 1.2)  
        return; // allow access  
}
```

Security Manager Trust Model

- Trust model has become increasingly refined over time
- JDK 1.0.2
 - Untrusted = any applet
 - Trusted = application
- JDK 1.1
 - Untrusted = unsigned applet
 - Trusted = signed applet or application
- **JDK 1.2 and beyond**
 - Many “shades of gray” in between



Java 2 Security Policy

- Identity
 - In Java 2, there are two identity-defining characteristics:
 - **Origin** (where the code comes from) and
 - **Signature** (who vouches for it)
 - Origin and Signature are represented with `java.security.CodeSource`
- Permissions (`java.security.Permission`)
 - Permissions include:
 - `java.io.FilePermission` for file system access
 - `java.net.SocketPermission` for network access
 - `java.lang.PropertyPermission` for Java properties
 - `java.lang.RuntimePermission` for access to runtime system
 - `java.security.NetPermission` for authentication
 - `java.awt.AWTPermission` for access to graphical resources
 - An example of a file permission:

```
p = new FilePermission("/applets/tmp/scratch", "read");
```

Java 2 Security Policy

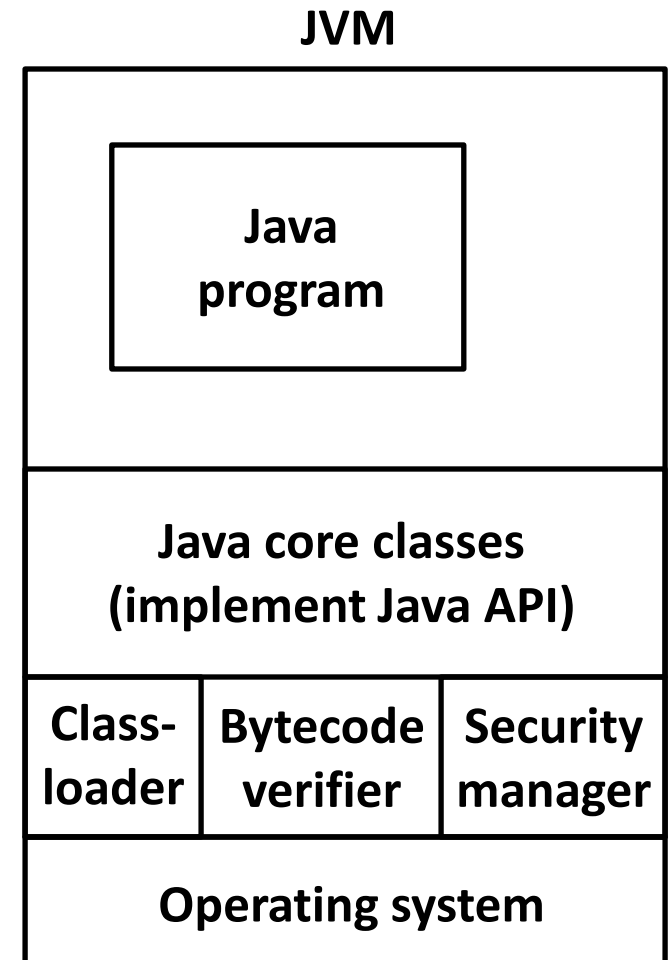
- Policy: A mapping from identity to permissions
 - The policy object is a runtime representation of policy usually set up by the VM at startup time (much like the Security Manager).
E.g.,

```
grant CodeBase "https://www.rstcorp.com/users/gem",
    SignedBy "*" {
    permission java.io.FilePermission "read,write",
        "/applets/tmp/*";
    permission java.net.SocketPermission "connect",
        "*.rstcorp.com";
};
```

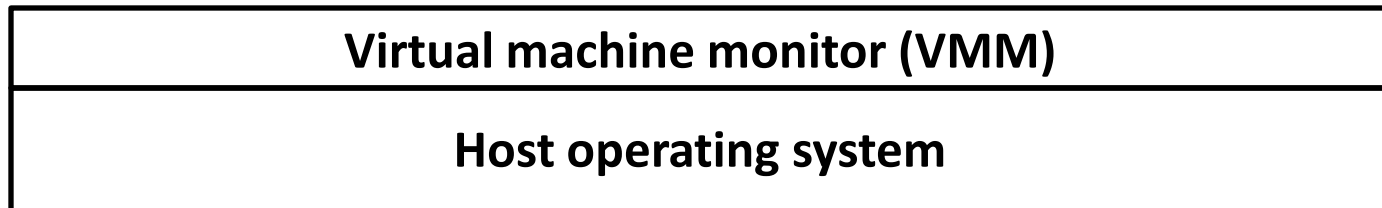
- Sun Java 2 policy can be set by users (bad idea) or system administrators
- Permissions are additive

JVM Wrap Up

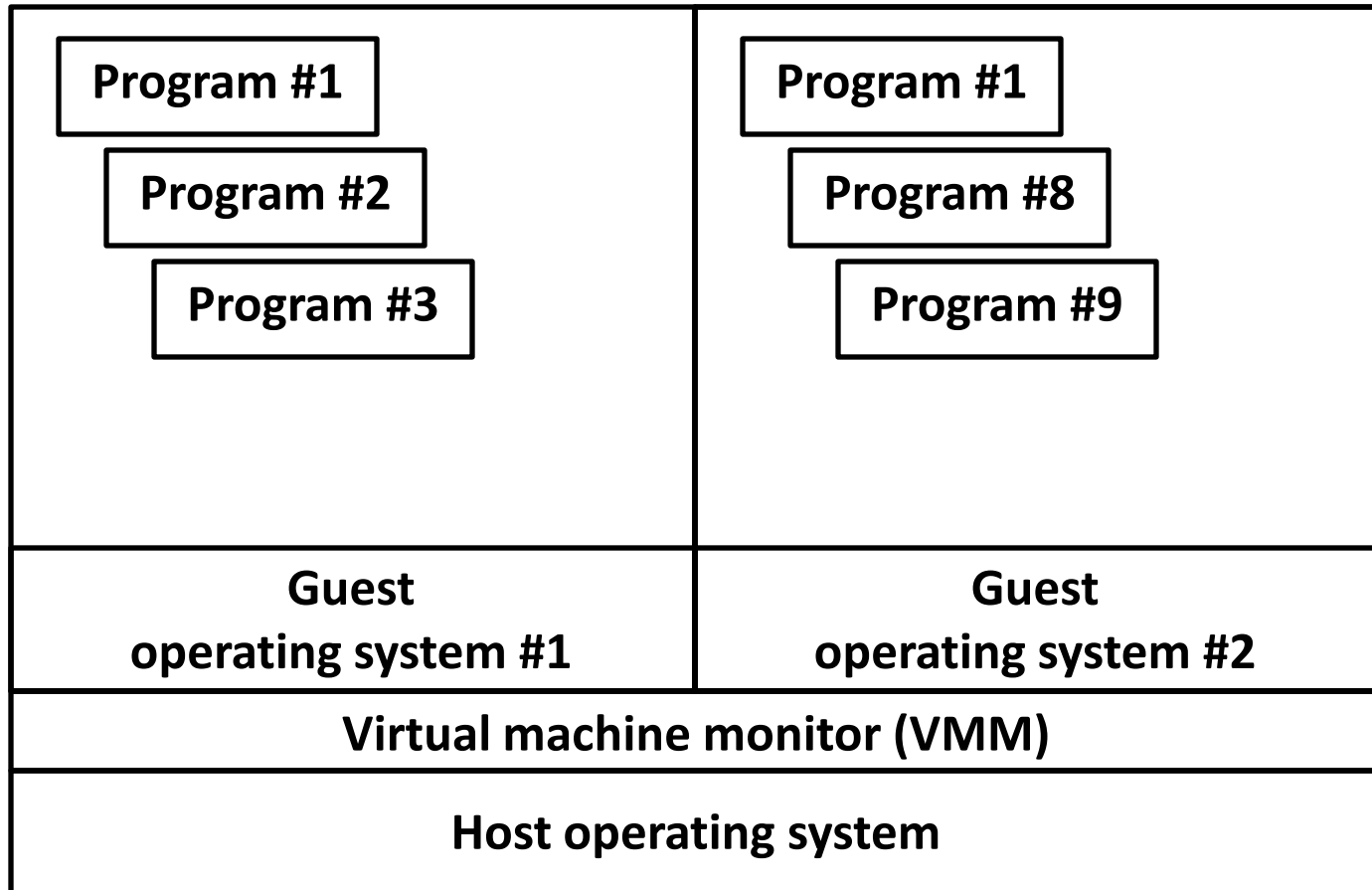
- Type safety provides memory protection
 - Between Java programs in a VM
 - Between Java programs and VM itself
- Security manager controls how Java programs interact with resources outside VM
 - Different programs need different levels of privilege
 - Non-trivial to decide which accesses to protected resources should be allowed
 - Hard to tell who is asking and why



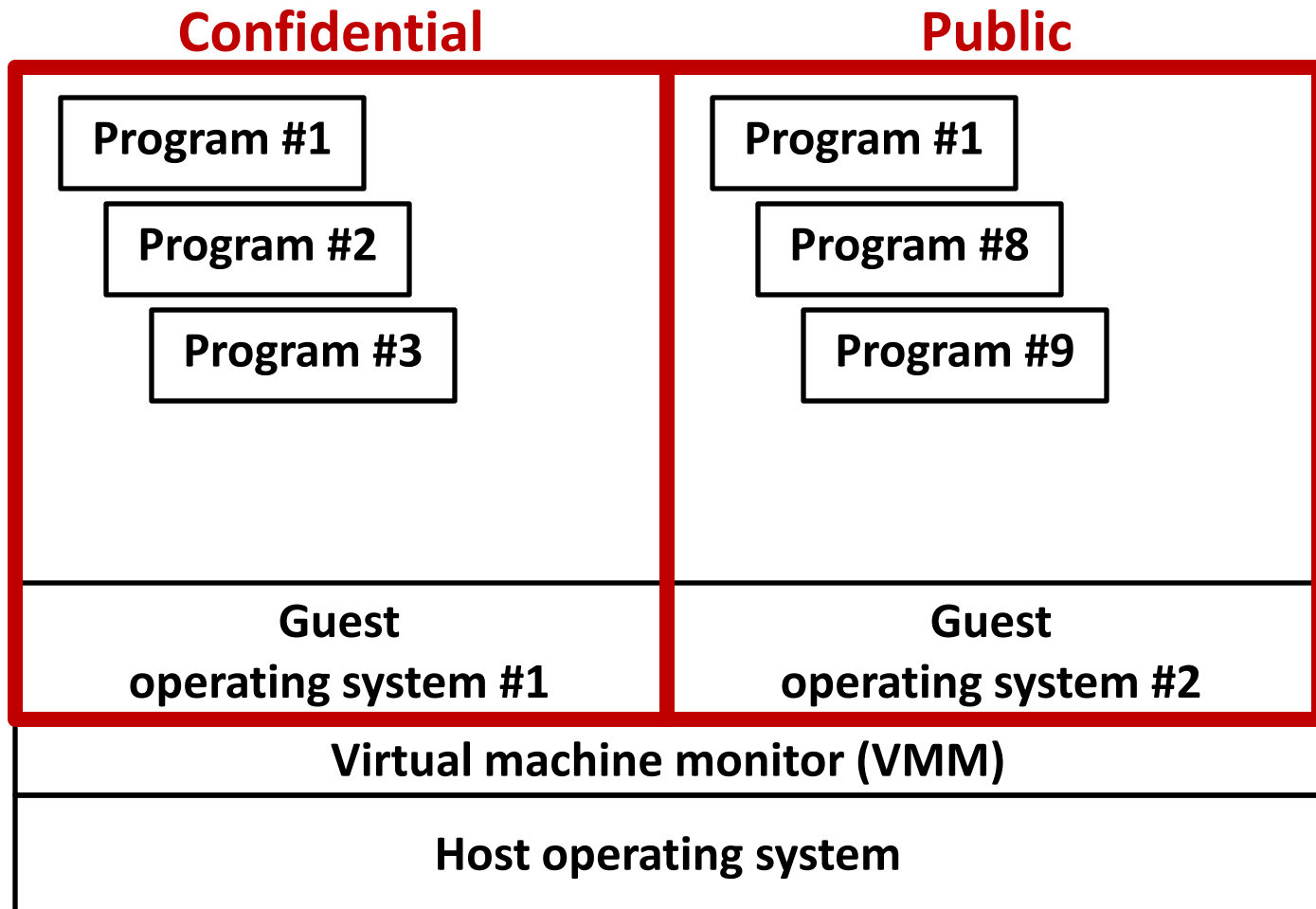
Sandbox #2: Virtual Machine Monitors



Sandbox #2: Virtual Machine Monitors



Sandbox #2: Virtual Machine Monitors



- Typical goal: complete isolation between guest OSes
- E.g., NSA's NetTop (SELinux + VMWare + Windows)

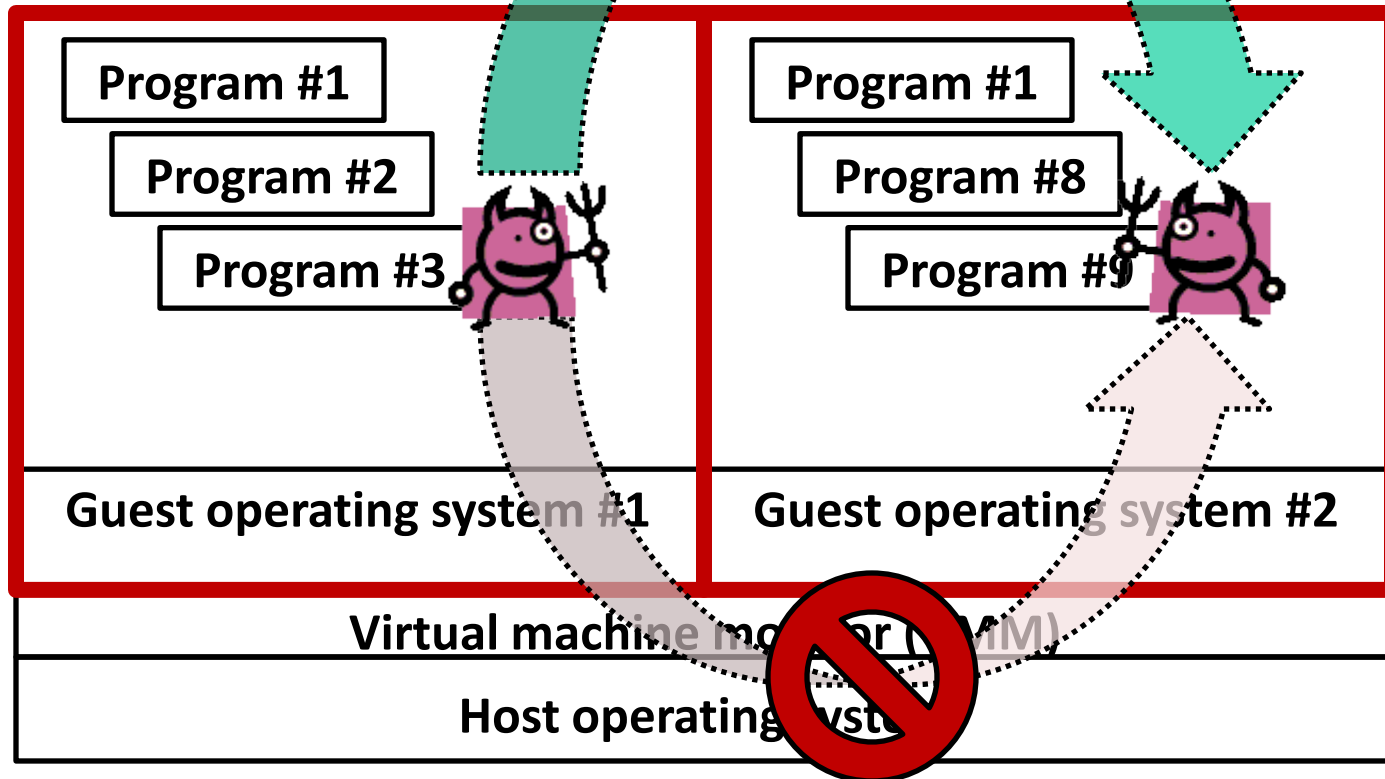
VMM Security Assumptions

- VMM assumptions
 - Malware can infect guest OS and guest apps
 - Malware cannot escape from the infected VM
 - Cannot infect Host OS
 - Cannot infect other VMs on the same hardware
- Requires that VMM protect itself and is not buggy
 - VMM is much simpler than full OS

Example Problem #1

Covert Channels

- Covert channel: unintended communication channel between isolated components
 - Can be used to leak classified data from secure component to public component



Example Problem #1

An Example Covert Channel

- Both VMs use the same underlying hardware
- To send a bit $b \in \{0,1\}$ malware does:
 - $b=1$: at 1:30.00am do CPU intensive calculation
 - $b=0$: at 1:30.00am do nothing
- At 1:30.00am listener does a CPU intensive calculation and measures completion time
 - Now $b = 1 \Leftrightarrow \text{completion-time} > \text{threshold}$
- Many covert channels exist in running system
 - File-lock status, cache contents, interrupts, ...
 - Very difficult to eliminate

Example Use #1

Intrusion Detection / Anti-virus

- Runs as part of OS kernel and user space process
 - Kernel root kit can shutdown protection system
 - Common practice for modern malware
- Standard solution: run IDS system in the network
 - Problem: insufficient visibility into user's machine
- Better: run IDS as part of VMM (protected from malware)
 - VMM can monitor virtual hardware for anomalies
 - VMI: Virtual Machine Introspection
 - Allows VMM to check Guest OS internals

Example Use #1

Sample Checks

- Stealth malware
 - Creates processes that are invisible to “ps”
 - Opens sockets that are invisible to “netstat”
- 1. Lie-detector check
 - Goal: detect stealth malware that hides processes and network activity
 - Method:
 - VMM lists processes running in GuestOS
 - VMM requests GuestOS to list processes (e.g., ps)
 - If mismatch, kill VM

Example Use #1

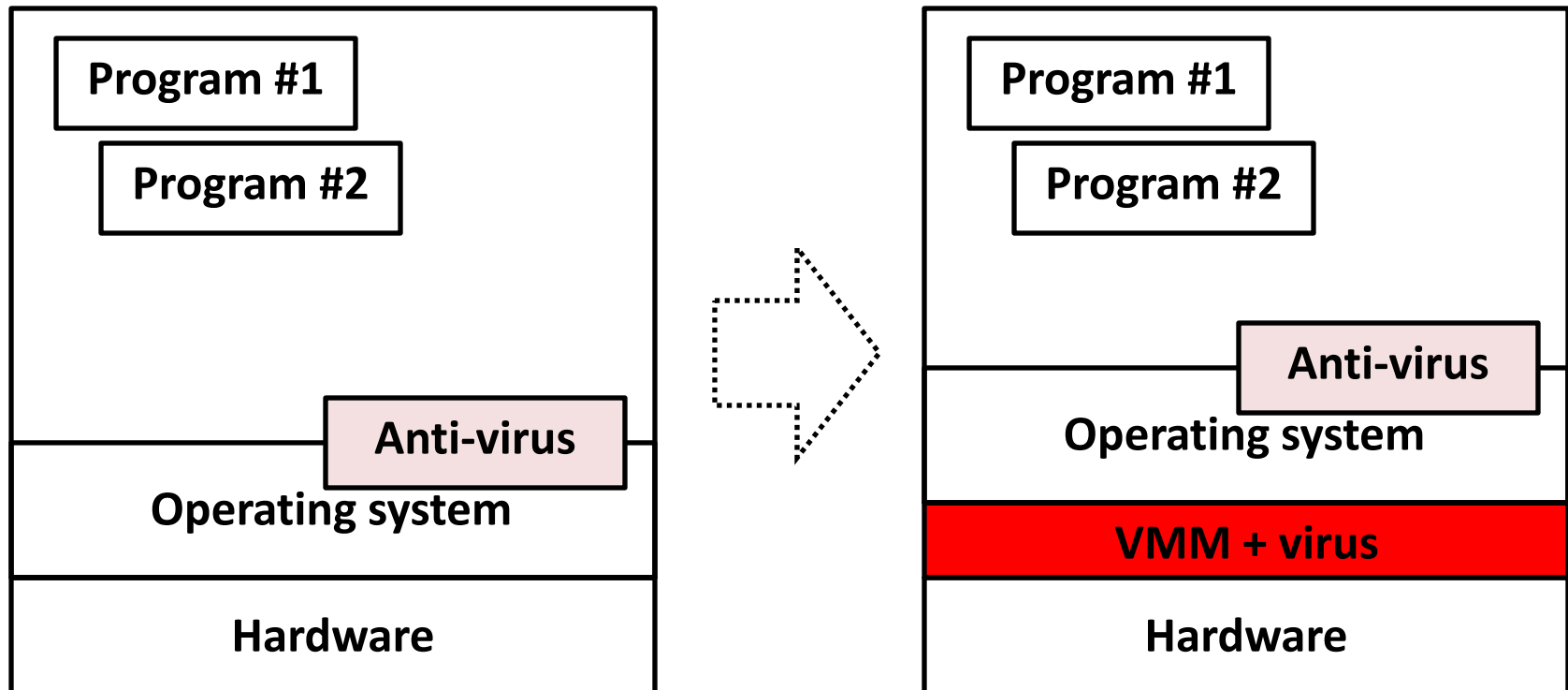
Sample Checks

2. Application code integrity detector
 - VMM computes hash of user app-code running in VM
 - Compare to whitelist of hashes
 - Kills VM if unknown program appears
3. Ensure GuestOS kernel integrity
 - Example: detect changes to `sys_call_table`
4. Virus signature detector
 - Run virus signature detector on GuestOS memory
5. Detect if GuestOS puts NIC in promiscuous mode

Example Problem #2

VM Based Malware

- Virus idea:
 - Once on the victim machine, install a malicious VMM
 - Virus hides in VMM
 - Invisible to virus detector running inside VM



Example Problem #2

VM Based Malware

- VMBR: a virus that installs a malicious VMM (hypervisor)
- Microsoft Security Bulletin, Oct 2006:
<http://www.microsoft.com/whdc/system/platform/virtual/CPUVirtExt.mspx>
 - Suggests disabling hardware virtualization features by default for client-side systems
- But VMBRs are easy to defeat
 - A guest OS can detect that it is running on top of VMM

VMM Detection

- Can an OS/application detect it is running on top of a VMM?
- Applications:
 - Virus detector can detect VMBR
 - Normal virus (non-VMBR) can detect VMM
 - refuse to run to avoid reverse engineering
 - Software that binds to hardware (e.g., MS Windows) can refuse to run on top of VMM
 - DRM systems may refuse to run on top of VMM

VMM Detection

- VM platforms often emulate simple hardware
 - VMWare emulates an ancient i440bx chipset
 - ... but report 8GB RAM, dual Opteron CPUs, etc.
- VMM introduces time latency variances
 - Memory cache behavior differs in presence of VMM
 - Results in relative latency in time variations for any two operations
- VMM shares the TLB with GuestOS
 - GuestOS can detect reduced TLB size
- ... and many more methods

VMM Wrap Up

- VMs can help provide confinement/isolation
- But...
- ... the perfect VMM does not exist
- VMMs today (e.g., VMWare) focus on:
 - Compatibility: ensure off the shelf software works
 - Performance: minimize virtualization overhead
- VMMs do not provide transparency
 - Anomalies reveal existence of VMM

Sources

- Some slides from L. Bauer and M. Reiter's 18-730 (CMU, Fall 2007)
- Some slides from D. Boneh and J. Mitchell's CS 155 (Stanford, Spring 2008)
- T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the Symposium on Network and Distributed System Security*, 2003.
- G. McGraw and E. Felten. *Securing Java*. John Wiley & Sons, 1999.