Operating System Security - Malicious Programs and Encapsulated Environments
CS239
Advanced Operating Systems
May21, 1997

Outline

- Malicious programs
- Encapsulated environments - the Java solution and others

Malicious Programs

Clever programmers can get software to do their dirty work for them

Programs have several advantages for these purposes

- Speed
- Mutability
- Anonymity

Kinds of Malicious Programs

- Trojan horses
- Trapdoors
- Logic bombs
- Worms
- Viruses

Trojan Horses

Seemingly useful program that contains code that does harmful things

Unsuspecting users run the Trojan horse to get the advertised benefit

At which time the Greeks spring out and slaughter your system

Particularly dangerous in compilers

Trapdoors

A secret entry point into an otherwise legitimate program

Typically inserted by the writer of the program

Most often found in login programs or programs that use the network

But also found in system utilities

Logic Bombs

Like trapdoors, typically in a legitimate program

A piece of code that, under certain conditions, “explodes”

Also like trapdoors, typically inserted by program authors

Worms

Programs that seek to move from system to system

Making use of various vulnerabilities

Other malicious behavior can also be built in

The Internet worm is the most famous example

Can spread very, very rapidly

Viruses

A program that can infect other programs

Infected programs in turn infect others

Along with mere infection, Trojan horses, trapdoors, or logic bombs can be included

Like worms, viruses can spread very rapidly

How Do Viruses Work?

When a program is run, it typically has the full privileges of its running user

Including write privileges for some other programs

A virus can use those privileges to replace those programs with infected versions

Typical Virus Actions

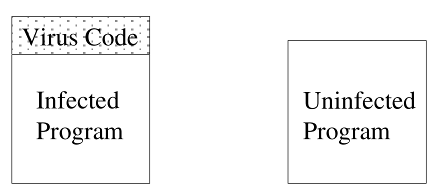
1). Find uninfected writable programs

2). Modify those programs

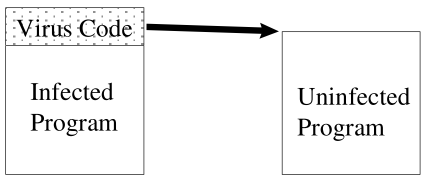
3). Perform normal actions of infected program

4). Do whatever other damage is desired

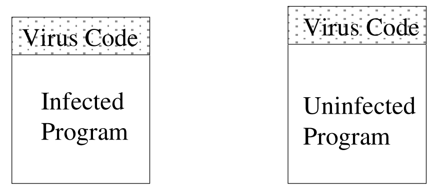
Before the Infected Program Runs



The Infected Program Runs



Infecting the Other Program



How Do Viruses Fit Into Programs?

- Prepended
- Postpended
- Copy program and replace
- Cleverly fit into the cracks

Some viruses take other measures to hide modifications

Where Can Viruses Occur?

Anywhere that imported code gets executed

- Imported programs
- Some inclusions in mail messages
- Boot sectors and other executable portions of media
- Macros attached to some data files

Dealing With Viruses

- Prevention of infection
- Detection and eradication
- Containment

Preventing the Spread of Viruses

Don’t import untrusted programs

But who can you trust?

Viruses have been found in commercial shrink-wrap software

Trusting someone means not just trusting their honesty, but also their caution

Other Prevention Measures

- Scan incoming programs for viruses
  - Some viruses are designed to hide
- Limit the targets viruses can reach
- Monitor updates to executables carefully
  - Requires a broad definition of "executable"

CS 239, Spring 1997 — Lecture 15 Page 19

Virus Detection

Many viruses have detectable signatures

But some work hard to hide them

Smart scanners can examine programs for virus-like behavior

Checksums attached to programs can detect modification

If virus smart enough to generate checksum itself, digitally sign it

CS 239, Spring 1997 — Lecture 15 Page 20

Virus Eradication

Tedious, because you must be thorough

Restore clean versions of everything

Take great care with future restoration of backups

CS 239, Spring 1997 — Lecture 15 Page 21

Containment

Run suspect programs in an encapsulated environment

Limiting their forms of access to prevent virus spread

Requires versatile security model and strong protection guarantees

CS 239, Spring 1997 — Lecture 15 Page 22

Imported Code and Encapsulated Environments

If so many bad things can happen when I run a program, I need to be very careful which programs I run

But that runs directly counter to industry trends

CS 239, Spring 1997 — Lecture 15 Page 23

Why Will You Import Lots of Executables?

- Java and other Web technology
- Trend towards building programs from components
- Active networks

CS 239, Spring 1997 — Lecture 15 Page 24

Access Control and Untrusted Executables

Is high quality access control enough?

Simply run each executable in its own protection domain

Carefully specifying accesses allowed

Then rely on the underlying access control to keep you safe

Why Aren't Standard Access Control Methods the Answer?

Programs execute under the user's identity, with the user's privileges

So the evil program has full user privileges

Most popular operating systems do not offer simple ways to limit the security domain of a program

Why Not Use More Flexible Security Roles, Then?

Alter Windows, Windows NT, Linux, Solaris, etc. to have flexible security domains

Possible, but -

- What access is allowable?
- How does it get set?
- How fast can you create the domains?

Another Problem With Standard Access Controls

Security domains are usually implemented as address spaces

So virtual memory can provide protection

But cross-address space communication is expensive

Problem if you're building programs out of components

Types of Solutions

- Improved OS access control
- "Padded cells"

Improving Access Control in Operating Systems

Most popular systems aren't built to handle more generalized access control

With domains not the same as process spaces

So we need new systems, or retrofits of old ones

Security retrofits rarely work well

Padded Cell Approaches

Control the list of things an untrusted piece of code can name

At a granularity below the address space

Essentially, execute the program in an encapsulated environment

CS 239, Spring 1997 Lecture 15 Page 31

Approaches to Building Encapsulated Environments

- Augmenting the OS
  - Solves the general problem
- Virtual machine and language-based approaches
  - Most suitable for downloading small executables
- Software-enforced fault isolation
  - Most suitable for composition of executables

CS 239, Spring 1997 Lecture 15 Page 32

Virtual Machine and Language Approaches

Define a virtual machine that does not allow "insecure" operations

Write all "untrusted" programs in a language that works on that virtual machine

Run imported programs through an interpreter for that language

CS 239, Spring 1997 Lecture 15 Page 33

The Java Solution

Java does precisely that

The Java virtual machine is meant to provide a secure execution environment

Also portable

The Java language ensures that all program operations are in the context of that VM

CS 239, Spring 1997 Lecture 15 Page 34

Security Concerns of the Java Approach

- Secure definition of virtual machine and language
- Secure implementation of virtual machine
- Secure implementation of language compilers/interpreters

CS 239, Spring 1997 Lecture 15 Page 35

Language and Virtual Machine Definitions

All security depends on the virtual machine not allowing insecure things

And on the language only working on the real machine through the virtual machine

So they must be carefully defined to not allow any insecure operations

CS 239, Spring 1997 Lecture 15 Page 36

Secure Implementation of the Virtual Machine

Given that the definition of the virtual machine is secure, we must be sure that the implementation matches the definition

Essentially, this is the same problem as verifying that an OS is secure

Perhaps on a smaller scale, though

Secure Implementation of the Language

The virtual machine is not the real machine

Other executables must work with the real machine

So something must ensure that the untrusted implementation only works through the virtual machine

Format of Imported Programs

Java cannot allow importation of compiled code

How can the local machine know that compiled code only calls the trusted virtual machine?

So "source" code must be imported

Then what?

Java Interpreters

You've got "source" code that must be executed

What now?

Run it through a trusted interpreter

Which verifies that only approved Java virtual machine operations are invoked

But interpreters produce inefficient code

"Just in Time" Java Compilers

Import the source, but compile it, instead of interpreting it

Compile performed on download or just before execution

Compiler must check security of code

Clearly a win if you run it more than once

Maybe a win even if only once

What is the Source Code?

Could be genuine source

But genuine source is expensive to compile/interpret

In Java, you can send "bytecode"

A portable format that is guaranteed to conform to the virtual machine

Essentially, a secure assembly language for that virtual machine

Access Control and the Java Virtual Machine

Fine, so the imported Java programs can only access the Java virtual machine

What can that virtual machine do?

A key issue, because all the security is based on the virtual machine operations being acceptable

Java Virtual Machine Capabilities

Basically, it doesn't allow much

- Very limited file access
- No process creation
- Very limited network communications
- Very limited examination of details of the host computer

Java's Dancing Pig Problem

All Java is good for is displaying dancing pigs on your computer's screen

Dancing pigs are of limited utility

What happens if you really need to write a file, create processes, communicate to a third party?

Previous Java Security Problems

Based on the vulnerabilities we discussed earlier

- Bug in verifier allowed applets to delete files
- Applets could generate and run raw machine code
- Allowed access to other machines in some cases
- Bunches of others - www.javasoft.com:81/sfaq

Software-Enforced Fault Isolation

Another approach to encapsulated environments

The virtual machine approach is limiting

Only operations permitted by virtual machine are allowed

Usually, only one language is supported

Building Programs Out of Components

Increasingly, programs are being built out of pre-written components

Due to COM, CORBA, etc.

So to build a program, slap together half a dozen pre-existing pieces

And add a little of your own code

But can you trust the pieces?

Performance Implications of Security Domains

If you build a program out of components, they need to interact efficiently

Security domains built on virtual address spaces require IPC to communicate

Can you afford to have your program's linked list code in a different address space?

Granularity of Security Domains

Existing systems provide security domains at the granularity of address spaces

That's too coarse

Composable programs need multiple security domains in one address space

Access Control Implications of Finer Granularity

Within a single address space, we need multiple access control domains for file references, IPC, etc.

But we also need access control for memory references!

Can no longer rely on hardware virtual memory protection

Other Approaches to Protecting Memory

- Segment matching
- Address sandboxing

The Basic To Be Solved

- Two mutually distrusting code segments share a single address space
- They export operations to each other
- How can we guarantee that they touch each other only through those interfaces?
- Given that they can issue each other's addresses

Other Constraints

- Must not be limited to a single language
  - Any executable must work
- Must be enforced at run time

Segment Matching

Examine executable about to be loaded for "unsafe instructions"

What is unsafe?

- Any jump or store to address that can't be statically verified
- E.g., jump through register, store through register

CS 239, Spring 1997 — Lecture 15 Page 55

Handling Unsafe Instructions

Define virtual memory segments that a piece of code can legitimately address

For each unsafe instruction, insert new instructions in the executable to check it at run time

Could be done at compile time or load time

CS 239, Spring 1997 — Lecture 15 Page 56

Checking Unsafe Instructions

Fundamentally, examine the non-static address the code proposes to use

If it's within the code's boundaries, let it happen

If not, prevent it

And report the violation

CS 239, Spring 1997 — Lecture 15 Page 57

Costs of Segment Matching

- Must reserve several registers for this purpose
  - Four, in Berkeley implementation
- Additional instructions performed
  - Four, in Berkeley implementation for a typical RISC processor

CS 239, Spring 1997 — Lecture 15 Page 58

Address Sandboxing

Reduces the cost of providing this level of safety

But loses ability to pinpoint attempts to bypass the security

Essentially, instead of checking, just apply a mask to unsafe addresses

Mask ensures that address is within permitted segments

CS 239, Spring 1997 — Lecture 15 Page 59

When Is Software-Enforced Fault Isolation Valuable?

- It's expensive, because it adds instructions to code
  - Perhaps in common cases
- But not nearly as expensive as IPC
- So it wins if the code performs a lot of IPC
- Also requires fast RPC across protection domains

CS 239, Spring 1997 — Lecture 15 Page 60