Mobile Code Safety

Dave Eckhardt
de0u@andrew.cmu.edu

Synchronization

- Checkpoint 2 megabyte bonanza
- No class Friday
- Book reports Friday midnight

Today's Lecture

- Safety for “Mobile Code”
- Includes 18.7
  - Among other things!!

Outline

- Motivation
- Careful hardware
- Careful interpreter
- Trusted compiler / Signed code
- Byte code verifier
  - Security policies
- Software Fault Isolation
- Proof-Carrying Code

Mobile code?

- Code from “somewhere else”
  - Java applet
  - Morris Internet worm
  - Melissa virus
- Is this a feature or a bug??
- ***Useful*** mobile code
  - PostScript / PDF
  - NeWS “pie menus”

Pie Menus

- Intuitive for users
- Not popular with window system providers
- Can random users ***replace*** default menus with pies?
  - NeWS: yes
- catalog.com/hopkins/piemenus

Network Tracing

- Who has run tcpdump/snoop/ethereal/...?
- How do they work?
  - Can we afford to copy every packet to user space?

Packet Filters

- Concept
  - Tell network stack ***which*** packets you want
    - tcpdump host piper.nectar.cs.cmu.edu
  - Network stack copies them to user memory
    - Just the headers? Part of the body?
- Approach
  - tcpdump writes a packet filter program
  - Network stack runs it for each packet
  - Program does return(nbytes) or return(0)

Packet Filter Example

```
(000) ldh  [12]
(001) jeq  #0x800        jt 2    jf  6
(002) ld   [26]
(003) jeq  #0x8002c250   jt 12   jf  4
(004) ld   [30]
(005) jeq  #0x8002c250   jt 12   jf 13
(006) jeq  #0x806        jt  8   jf  7
(007) jeq  #0x8035       jt  8   jf 13
(008) ld   [28]
(009) jeq  #0x8002c250   jt 12   jf 10
(010) ld   [38]
(011) jeq  #0x8002c250   jt 12   jf 13
(012) ret  #68
(013) ret  #0
```

Packet Filter Issues

- tcpdump loads filter program into OS!
- Is this ok?

```
(010) ld   [1048576]
```

- How about this?

```
(010) ld   [-50]
```

- How about a ***real*** program?

```
(000) ldh  [12]
(001) jeq  #0x800          jt 1     jf  1
```

Packet Filter Restrictions

- Abstract machine, not real instructions
  - Must be run by interpreter
- Addresses are range-checked
- Small scratch-pad memory for program use
- No loops!
  - Can't checksum an arbitrary-length packet

Packet Filter History

- History
  - J. C. Mogul, R.F. Rashid, and M.J. Accetta, The packet filter: An efficient mechanism for user-level network code. SOSP 11 (1987) http://citeseer.nj.nec.com/mogul87packet.html
  - Xerox Alto (1976) (single address space)

Careful Hardware

- Approach
  - Define safe & unsafe behaviors
  - Embed police function in hardware
- Example
  - Hardware virtual memory / memory protection
  - Clock interrupts
  - Multics/Hydra/CAP/EROS

“Careful Hardware” Issues

- Context-switch overhead
  - Switching to user mode (and back)
  - Too expensive for every packet
- Hardware protection doesn't cover software issues
  - Can Steve's fancy packet filter delete my files

Careful Interpreter

- Approach
  - Run each instruction “by hand”
  - Enforce safety policy
- Example
  - Packet filters
  - JavaScript (uh-oh)

“Careful Interpreter” Issues

- Requires special language / abstract machine
- Can be ***very*** slow
- Hard to get safety policy right
  - People often focus on features
    - Pop up a nice temporary dialog box
    - Specify window position, stacking
  - Hard to add good rules later
    - Don't allow pop-behind ads...
    - Allow only finite loops...

Trusted Compiler

- Approach
  - Language designed for safety
    - No “pointer arithmetic”
    - Automatic memory management
  - Compiler rejects code violating safety policy
  - Compiler contains no bugs
- Example
  - ML, Modula-3

“Trusted Compiler” Issues

- Executable ***really*** from trusted compiler?
  - Compile before every execution?
  - Code signing (see below)?
- Compiler ***really*** contains no bugs?
  - Certainly not in the optimizer!!
- Language-embedded safety policy is ***very*** static!
  - Probably ignores many concerns
  - Very hard to adjust afterward

Signed Code

- Intuition
  - Too hard to verify code is safe
  - Too hard to ***specify*** safety policy
  - Surely Microsoft is careful and honest?
    - s/Microsoft/Andrew Systems Group/

Signed Code

- Approach
  - $TRUSTED_ORG builds program
    - Safe design, language, programmers, compiler
    - No last-minute viruses in QA group!
  - $TRUSTED_ORG digitally signs program
    - Or ***at least*** puts fancy holograms on the CD
  - Code consumer verifies signature
    - Ok to run it!
- Example: Microsoft ActiveX

“Signed Code” Issues

- Good news
  - Supports any source language
    - We can certify C++ code!
  - No restrictons on executable performance
- Bad news
  - Microsoft signs ***everything they write***
    - Outlook, IE, IIS, ...
  - So does/would Sun, Red Hat, OpenBSD...
- Better than nothing...

Byte Code Verifier

- Approach
  - Allow any compiler, any author
  - Require abstract machine code
  - Scan program before execution
    - “Prove” code is safe
  - Compile abstract code to real machine code?
  - Verify certain operations during execution
- Example: Java

Byte Code Verifier - Checks

- Class file well formed?
  - Correct magic number
  - No extra/missing bytes
  - File parses successfully
- Class is “sane”
  - Every class has a real superclass
  - “Final” classes/members are not overridden

Byte Code Verifier - Dataflow

- Concept
  - Stack-based virtual machine
  - Scan every instruction
    - ...every way it's reachable
- Checks @ each instruction
  - Stack state the same (size, #objects)
  - Register accesses type check
  - Type check: operators, calls, assignments
- http://java.sun.com/sfaq/verifier.html

Byte Code Verifier – Example

```
class HelloWorldApp {
  public static void main(String[]
args) {
    String s;
    if (args.length < 1)
      s = "Hello World!";
    System.out.println(s);
  }
}
```

- println() call reachable via two paths, one bad

Byte Code Verifier

- Shouldn't the compiler catch that?
  - Yes (it does)
- So why must we verify it?
  - That's the whole point
- How good is this verifier?
  - Occasional bugs
  - Limited proving power
  - Only low-level safety

Limited Proving Power

```
class HelloWorldApp {
  public static void main(String[]
args) {
    String s;
    if (args.length < 1)
      s = "Hello World!";
    if (args.length < 1)
      System.out.println(s);
  }
}
```

- Still fails

Only Low-level Safety

- Verifier provides “language safety”
  - Can't step outside type system
  - Can't crash virtual machine
  - JIT optimizer can depend on initialized variables
  - Lots of good stuff
- Doesn't address ***program-level*** safety
  - Opening files, accessing network

Higher-level Issues

- Initial approach
  - "Applets can't access the file system *at all*"
  - Can't even save preferences
- "Applets signed by corporate IT ***can*** access files"
  - The best you can do for arbitrary programs?
  - Doesn't address web applets
- Pluggable security policies
  - "Applets can access files in $HOME/.prefs/$classname"

Pluggable Security Policies

- Load class into JVM
  - Via verifier, so it's sort-of-ok
- Class associated with protection domain
  - Depending on class loader, signatures, ...
- Protection domain contains ***Permission*** patterns
- Before doing something dangerous

```
FilePermission perm = new
FilePermission("/temp/testFile",
"read");
AccessController.checkPermission(perm)
;
```

Pluggable Security Policies

- checkPermission() examines stack
  - Class can “donate” protection domain to callees
    - doPrivileged { ... }
  - Otherwise, consult default domain
- Text 18.7
- API docs for java.security.AccessController
- Where do permissions come from????
  - research.sun.com/research/techrep/2002/smli_tr-2002-108.ps

Software Fault Isolation

- Goals
  - Want to run ***real*** machine code
    - No byte code
    - No restrictive stack scanner
  - Focus: memory safety
    - Don't read unauthorized memory
    - Don't write unauthorized memory

Software Fault Isolation

- Approach
  - ***Edit*** machine code before execution
    - load instruction?
      - if ((address < ...) && (address > ...) load ...
    - same for store
  - Optimize out redundant checks
  - Resulting code ***must*** be memory-safe
    - Even if original code wasn't!

“Software Fault Isolation” Issues

- Execution can be slow
  - Especially if you check loads (!!)
  - Especially if code accesses multiple memory regions
  - Especially if memory regions aren't power-of-2 size
  - ...
- Implements ***only*** memory safety
  - Not: “read table entry only if valid bit on”

Software Fault Isolation

- Wahbe, Lucco, Anderson, & Graham. Efficient software-based fault isolation. SOSP 14 (1993)

Proof-Carrying Code

- Goals
  - Want to run real ***unedited*** machine code
  - Want to support more than memory safety
    - "valid bit" example
    - Mutex is acquired before ...
    - Mutex is released after ...
    - Execution time limit

Proof-Carrying Code

- Publish a safety policy
  - Safe invocation of each instruction
  - Precondition (which memory is readable...)
  - Postcondition (termination, ...)
  - Safety predicate generator
- Code comes with proof of safety predicate
  - Generated by code author - somehow
  - Attached like a symbol table

Proof-Carrying Code

- Running code
  - Predicate generator produces ***safety predicate***
    - Instruction 0 is safe because it's an ADD
    - Instruction 1 is safe because it reads from safe memory
    - Instruction 2 is safe because it stores to safe memory
  - Verify that alleged proof actually proves predicate
  - Safe to run the code!

Proof-Carrying Code

- Where do proofs come from???
  - Consumer ***publishes*** safety predicate generator
    - So it's clear what must be proven
  - Compiler may be able to produce proof
    - Maybe with user assistance
  - User can hand-write proof
    - Especially for hand-written assembly language
- What code is mutated?
  - Proof no longer valid... (or no harm done!)

“Proof-Carrying Code” Issues

- Good news
  - Can run “optimal” code
  - Verification “typically” fast
- Bad news
  - Automatic proof generation is hard
  - No library of agreed-upon safety policies
  - Proof verification can be expensive
    - Naively, a ***gigabyte*** of memory!

Summary

- Performance?
  - Time to start running
  - Throughput while running
- Safety policy
  - Static (language designer)
  - Dynamic (system administrator)
- Trust model
  - Trust people, programs, or proofs?

Summary

- Careful hardware – could EROS make it happen?
- Careful interpreter – ok for slow applications
- Trusted compiler / Signed code – dubious
- Byte code verifier – has its uses, limits
- Software Fault Isolation – hmm...
- Proof-Carrying Code – hmm...