

Resource Access Control

Brian Carrier
CS 590U

Outline

- Definition & Motivation
- Overview
- Policy Creation
- Policy Enforcement
- Policy Language
- Breaking the Systems
- My Project

Question Policy

- Questions from any registered student or Prof. Li are unlimited and free
- Questions from unregistered students:
 - Question #1 is free
 - Questions #2 - 6 are \$0.05 each
 - Additional questions are \$5.00
 - Mahesh Special: 10 questions for \$19.99+tax
- Proceeds go to the new CS building fund.

Definition

- Resource access control limits what an application (process) can do
- In modern systems, system resources are controlled by system calls
- Resource access control typically applies a policy to system calls

Two Uses of RAC

- RAC is used for Host-based Intrusion Detection and Sandboxing.
- Both uses are similar, but have different enforcement mechanisms

Host-Based IDS

- The goal is to detect an application that has been compromised
- The policy defines the expected normal behavior of the application
- This system will allow access to the resources, but may generate alerts

Sandbox

- The goal is to confine an un-trusted application
- Applets and code from the Internet
- The policy defines the maximum that the application can access
- This system will prevent an access to the resources

3 Major Phases of RAC

1. Policy Creation
 2. Policy Enforcement
 3. Policy Language
- Existing work examines one or two of the phases
 - We will cover existing work in each of these areas

1. Policy Creation
2. Policy Enforcement
3. Policy Language

Policy Creation

- Most existing work does not address this phase
- Common Techniques:
 - Ignore it or wave hands in the air
 - Write it by hand
 - Train the enforcement tool
 - Static analysis of applications
 - Sundar will cover this in more detail

1. Policy Creation
2. Policy Enforcement
3. Policy Language

Enforcement Overview

- Research Goals:
 - Applications need not be modified
 - User-space vs. Kernel-space
 - Optional vs. Required
 - Real-time (sandbox) vs. Reaction (IDS)
- Main Concept: Monitor system calls

Audit Logs

- Solaris BSM will log system calls
- IDS systems can audit the logs to detect an incident
- Not appropriate for sandboxing
- Requires a trusted process to review logs
- May not be able to check arguments

Libraries

- Libraries can be modified to check permissions for system calls
- This can be bypassed by an attacker
- This technique is not really used

Custom Browser

- Useful in environments with distributed applications
- Mobile code is downloaded and sandboxed
- Similar to Javascript and Java executing in Netscape or IE
- Java Virtual Machine

Debugging Facility

- Many OS's offer a debugging method: ptrace() or '/proc/'
- A process attaches to the process being monitored and intercepts system calls
- Requires that the monitoring process run as 'root'
- Must re-attach to child processes after fork()

Kernel Modules

- Technique uses a loadable kernel module for each system call
- Kernel modules are executed in the kernel before the actual system call
- A user-space application maintains policies and allows for management

Kernel Module Benefits

- Cannot be bypassed (libraries)
- Requires no application or kernel modifications
- Does not require a monitoring application to run as 'root'
- Handles child processes easier

Kernel Modifications

1. Modify the kernel to check policies on system calls - same as modules
2. Modify how kernel works (i.e. Domain Type Enforcement)
3. Add new API so that parents can dynamically control the child's policy (Peterson et al.)

File System Drivers

- Prevalakis and Spinellis protect only the file system
- Special NFS server or Perl FS driver
- Policy dictates which files a process can read
- Can provide 'fake' files with only minimal data

Dealing with Violations

- Return a system call error
- Kill the application
- Fake the environment

1. Policy Creation
2. Policy Enforcement
3. Policy Language

Policy Language

- Languages have not been a focus of research in this area
- Languages range from simple statements to object oriented with methods
- Complexity Factors:
 - State-based
 - Arguments to system calls
 - Abstraction level

Mitchem, Lu, O'Brien

- Kernel Hypervisors:
 - <type> <identifier> <permissions>
 - Type: "file" | "socket" | "process"
 - Identifier: <path> | <IP> | <PID>
 - Permissions: "read" | "write" | "read/write" | "none"
 - socket "192.168.1.10" read/write
 - file "/etc/passwd" none
- Rules were per process and translated to each system call

Prevelakis, Spinellis

- Sandbox via new file system
- Protects file access only
- Can provide “fake” files
- Simple rules (per application):

```
R_X_  /bin/ls
R___  /etc/passwd      /blah/passwd-2
```

Ko, Fink, Levitt

- Audit Log Detection
- Set of rules per application

```
PROGRAM foo(U)
  exec ("/bin/ls");
  write ("/usr/spool/mail/" + U.name);
  read (*);
  ~read("/etc/passwd");
  write(F) :- inside(F, "/tmp");
END
```

Chari, Cheng

- Kernel modification enforcement
- Policies for objects, users, signals
- Don't give full grammar, only Apache example

Chari, Cheng contd.

- Files:

```
/home/foo/*.txt      rwc      644
/etc/*                r
```

- They say you can apply rules for mounting, sockets, IPC, UIDs, and EUIDs but they don't give the syntax

Provos

- Improvements on existing work - kernel and user enforcement
- Simple rules per system call:

```
native-open: filename eq "/etc/passwd"
  then fail
native-open: filename eq "/tmp" then
  permit
native-bind: sockaddr match "inet-*:23"
  then permit
```

Jaeger, Prakash

- Browser enforced
- Principal and Object Groups are defined
- Rights and Exceptions are defined:

```
{p_group, operations, o_group}
except: {p_group, operations, o_group}
```

Jaeger, Prakash contd.

- Scientists can read /etc:
`{sci, r, /etc}`
`except: {sci, r, /etc/passwd}`
- The interpreter can be specified in the principle group
`{sci:a.out, r, /etc/passwd}`

Goldberg, Wagner, Thomas, Brewer

- Debug enforcement - Janus
- Uses modules: basic, tcpconnect, path, ...
- Each module knows what system calls to monitor and how
- Rules can exist on a system, application, or user-level
`path allow read,execute /bin/ls`
`path deny read /etc/passwd`

Walker, Sterne, Badger, Petkac, Shermann, Oostendorp

- Domain and Type Enforcement
- Every object has a type
- An object can change domains when it executes a given program
- Each domain has resource access policies, an entrance command, and list of exit domains

DTE contd.

```
domain daemon_d = (/sbin/init),
    (rxd->binaries_t),
    (auto->login_d);
domain login_d = (/usr/bin/login),
    (rd->readable_t),
    (exec->user_d, admin_d);
assign readable_t /etc;
```

Ko, Ruschitzka, Levitt

- Distributed applications: audit logs
- Detect with traces of system calls
- If system call falls out of grammar then fail
- Local and global variables exist

Ko, Ruschitzka, Levitt

```
<foo> -> <valid_op> <foo>
<valid_op> ->
  open_r_worldread
  | creat_file
    { if !(Inside(P, "/tmp")) then
      violation; fi}
  | chmod
    {if !(Created(F)) then violation();
    fi}
```

Kain and Sekar

- Expand on Janus - Debug enforced
- Enforced with a monitoring process - C++ object with given base class
- Object defines methods to be called for given system calls:

```
read_entry(...);  
read_exit(...);
```

Kain and Sekar contd.

- Groupings can be used:

```
readCalls(f) ::= read(f), readdir(f)...  
readCalls_entry(f,...) {...}
```

Sekar, Brown, Segal

- Kernel and User-land enforcement
- Auditing Specification Language (ASL)
- Relies on event patterns
- Events:
 - sequential: p1; p2
 - repetition: p{n1,n2}
 - real-time constraints: p within [t1,t2]
 - Atomicity: nonatomic d in p

Sekar, Brown, Segal

- Can call external functions and structures

```
open (file) |  
((f = realpath(file)) &&  
 (f == "/etc/passwd")) -> fail (-1)
```

- Can contain suspect programs

Fraser, Badger, Feldman

- Kernel enforcement to wrap COTS
- C++ based, system call or path to monitor are registered
- Can have internal methods and variables

```
bsd::op {open} {  
    if ($path == "/etc/passwd") {  
        return WR_DENY;  
    }  
}
```

Breaking Them

- Wagner & Soto: Systems that rely only on sequence of system calls can be fooled if return value is not checked
- Tal Garfinkel: Problems of maintaining (duplicating) OS state and not monitoring other system calls that can achieve the same thing

The Project

- Examine languages in more detail
- What is needed?
- How expressive need they be?

Questions?