Splitting Interfaces – Making Trust Between Apps and OS Configurable
Trust Model for an Application

• Kernel is seldom exploited directly (eg. malicious system calls)
• Attacker takes control of kernel through privileged applications
• Application cannot trust the kernel or other applications
• But need to communicate with them
• Problem: How to separate an application securely (retaining confidentiality, integrity) yet not isolating it
Proposed Solutions

• Microkernels
  – Reduced code running in kernel
  – Doesn’t really change anything

• Fine Grained Access Control (eg. SELinux)
  – Complex policy
  – Processes still can wreak havoc on files accessible to them (eg: If ssh server is compromised, can leak keys)
Proposed Solutions

• Separate applications into own virtual machine
  – Eg. Terra – Applications into VMs
  – Problem: This separates applications, but also isolates them, forcing untrusted apps into TCB
  – Terra allows only network socket level communication
Key Insight

• Intuition:
  – The TCB of an application = Code that it executes + everything that this code depends on (library functions, system calls)

• Not necessarily the case

• Eg. Concept of “trusted wrappers”
  – Wrap untrusted calls with some functionality that detects any malicious attempt
  – Eg: Wrap HTTP by SSL
Solution

• Separate application from other apps/kernel
  – Use separate VM for app with a Private OS separate from Commodity OS

• Provide interaction between apps/kernel in a secure way
  – Application developer decides what is sensitive and what is not
    • Separate sensitive part into VM on Private OS
    • Public part remains on Commodity OS
    • Interaction between apps also passes through kernel (eg. pipe(), mkfifo())
      • Sensitive part communicates through system calls with other apps
  – Use policy to decide if system calls are to be performed on commodity OS or private OS
Summary of Goals

• Separate application from other applications/kernel
  – Each application runs in own VM

• Applications need to interact securely with other untrusted apps/kernel
  – Security-sensitive part executes in own VM
  – Allow minimal changes in application by separating through policy on system calls and not code
Architecture

• Private OS has Proxos library
  – Mimics system call interface
• Proxos routes non-security-sensitive system calls to commodity OS and security-sensitive system calls to private OS depending on policy (= routing rules)
• Security-sensitive part of application = private application (on Proxos)
• Non-security-sensitive part of application = host process (on Commodity OS)
Architecture
Working

• Developer specifies routing rules
• Private application + private OS methods + routing rules compiled = single binary loaded in VM
• To run a private application, user invokes host process on commodity VM
  – Loads binary into VM
  – Host process is the “identity” of private application on commodity OS (Access Control, Communication etc)
Security Guarantees of Architecture

• Assumption
  – VMM enforces separation
  – Application developer correctly specifies routing rules

• Guarantee
  – Confidentiality and integrity of sensitive private application data inspite of malicious commodity OS
    • VMM => No direct interference possible
    • Commodity OS can interfere with system calls routed to it, which are not security-sensitive
  – Availability not guaranteed
Routing Language

• Needs to specify which system calls go where (arguments need be considered)

• Solution: Partition system calls by resources they access
  – Disk, Network, UI, Randomness, System Time, Memory
  – Randomness, System Time -> Always routed to VMM
  – Memory -> Always routed to private OS
Example Routing

# Rules Section
# route accesses to /etc/secrets to private OS
DISK: ("/etc/secrets", priv_fs)
# route accesses to UNIX domain socket bound
# to /tmp/socket and TCP socket bound to peer
# 192.100.0.4 port 1337 to private OS
NETWORK: ("unix:/tmp/socket", priv_unix),
    ("tcp:192.100.0.4:1337", priv_tcp)
# route all accesses to stdin, stdout
# and stderr to private OS
UI: (*,priv_ui)

# Methods Section
# individual methods in the private OS
# that are bound to system calls
priv_fs = {
    .open = priv_open,
    .close = priv_close,
    .read = priv_read,
    .write = priv_write,
    .lseek = priv_lseek
}
Implementation

• Commodity OS -> Linux
• VMM -> Xen
• Modifications
  – Startup and shutdown of private applications
  – Forwarding system calls between VMs
  – Trusted path facility
Start-Up Process

• Private app should have only privileges of host process
  – Host process is identity of private app
• Commodity OS should not affect private VM creation
• Creation done by management VM (dom0)
Start-Up Process

Host Process: `pr_execve(app_name)`

Linux Kernel:
- Allocate shared mem
- Pass (app_name, addr of shared mem) to VMM
- Start private VM
- Associates VMID with PID of host for identifying future syscalls

VMM/dom0:
- VMM creates private VM, gives it shared mem addr
- Associate VMID with addr
- Return VMID
- Maps syscall shared mem into addr space

Private VM:
- When private VM gives map request, checks if VMID corresponds to addr
System Call Forwarding

• Proxos decides to forward a system call to commodity OS
• Writes to shared memory
• Linux OS reads from shared memory, gets VMID, associates with host process
• Pushes args for syscall onto stack
• Calls sys_call
• Note: No domain crossing in Linux OS
Trusted Path

• Why? User thinks she is running private app, when in fact malicious commodity OS controls the console

• Use VMM to get handle on Console, X
  – In Xen, use dom0 to get handle on xenconsole
  – Emulate X by attaching to xnest server on dom0
Proxos

• Use miniOS to implement
• Only single application runs => Single Address Space
• Basic memory, page table management
  – For private app, mapping shared page for syscalls
• Block driver
  – Accessing disk for files directly
• Console driver frontend
  – Trusted path
System Call Routing

• Find rule in policy and route accordingly
• Default: Route to commodity OS
• fork
• target PID in syscall args tells which host process associates As miniOS does not have concurrent process support, private app must be made sequential
• execve
  – If without fork, host process terminates and new program is host process
  – If with fork, parent retained as host process
System Call Routing

• select, poll
  – Proxos alternates between fds in private and commodity OS
Private OS Methods

• Extensions of syscall interface with specific functions in the private OS
  – Private File System
    • Most functionality exported to Linux OS, but maintains secrecy by encryption
    • Trusted wrappers
  – Trusted Path for X window messages and Console
Applications – Secure Web Browser

- Dillo, a graphical web browser
- Sensitive data read / write from disk is on private file system
- Dynamic linked library for “themes” in Linux OS could not be used
- Converted to static
Application – SSH Authentication Server

- Password file, host key and config files on private file system
- Native process listens for connection
- When connection comes in, fork and pr_execve() a child private app
- During authentication, target PID changed to child process in Linux OS
- After authentication (setup of pipe and start of command shell) target PID switched back to listening process in Linux OS
SSH Server

Private VM
- Private OS
  - Passwords
  - Host Keys
- SSH Server
  - Encrypt
- Proxos

Commodity OS VM
- Pipe
- Command Shell
- Host Process
- Linux Kernel

Network
SSH Server

Partitioning Interfaces to Resources

Partitioning Code (Provos et al)
## Cost of Porting

<table>
<thead>
<tr>
<th>Application</th>
<th>Rules</th>
<th>LOC Modified</th>
<th>Total LOC</th>
<th>% Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dillo</td>
<td>53</td>
<td>22</td>
<td>20,528</td>
<td>0.1%</td>
</tr>
<tr>
<td>SSH Server</td>
<td>35</td>
<td>108</td>
<td>27,000</td>
<td>0.4%</td>
</tr>
<tr>
<td>Apache &amp; OpenSSL</td>
<td>28</td>
<td>667</td>
<td>135,916</td>
<td>0.5%</td>
</tr>
<tr>
<td>Glibc</td>
<td></td>
<td>218</td>
<td>1,775,440</td>
<td>0.01%</td>
</tr>
</tbody>
</table>
Performance of System

• System call forwarding overhead
  – Context Switch Cost: 14us.
Performance of Applications on the System

- Compare private + public applications on Xen vs unmodified apps on Xen
- 0.35s to create new VM + 0.37s to start openSSH = 0.72s per SSH client and 15.7us for each system call
- Throughput increased (4.75 Mbps to 5.04 Mbps)
Take-Away

• Proxos allows application to specify what it trusts and does not in an OS

• Specifying trust by partitioning interfaces to resources results in intuitive policy description and minimum change by which an application can be split into sensitive and non-sensitive parts
• Is this general enough to allow two or more processes separated into private parts communicate with each other

• Is the expressive power of this policy same as can be achieved with code separation