Shifting GEARS to Enable Guest-context Virtual Services

September 18, 2012
We advocate hoisting implementations of VMM services up into the guest without guest cooperation.

GEARS (Guest Examination And Revision Services): framework for guest-context virtual services

Allows easy development of services, with potential performance gains and small increase in VMM complexity.

Two prototype guest-context virtual services
- Overlay networking accelerator (latency decrease by 3-20%)
- MPI Accelerator (native memcpy bandwidth for colocated VMs)
- Overview
- Motivation
- GEARS
- Evaluation of Tools
- Example Service
- Conclusions
VMM code running within the guest can be simpler, operates at a higher semantic level

Overheads from VMM exits are substantial

Allows new classes of services that wouldn’t be possible

Alternatives, e.g. paravirtualization, symbiotic virtualization, require guest cooperation

Need a bidirectional interface between VMM and guest, no guest cooperation
- Overview
- Motivation
- GEARS
- Evaluation of Tools
- Example Service
- Conclusions
PALACIOS VMM

- OS-independent, embeddable VMM
- Support for multiple host OSes (Linux, Kitten LWK)
- Open source, available at http://v3vee.org/palacios
We claim that to enable wide range of services, need 3 major tools
- System call interception: track userspace events
- Process environment modification: pass info to processes
- Code injection: run VMM code in guest (app and kernel)

These tools could be built in any VMM, and require little implementation effort
- Adds little complexity to VMM codebase
- Service developer provides implementations and GEARS transforms and places them appropriately in guest

<table>
<thead>
<tr>
<th>Component</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Call Interception</td>
<td>833</td>
</tr>
<tr>
<td>Environment Modification</td>
<td>683</td>
</tr>
<tr>
<td>Code Injection</td>
<td>915</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2431</strong></td>
</tr>
</tbody>
</table>
SYSCALL INTERCEPTION

- Introduce system calls as exceptional events to VMM
  - SYSCALL/INT 0x80

- Can build several services on top of this technique
  - Sanity check args against errors/attack
  - Match system call patterns to higher level events

- Used in GEARS to track user-space events at a fine granularity

- Either exit on all syscalls or be selective (requires injected module)
LSTAR

If (syscall_map[i])
    hypercall
else
    goto original

Original syscall entry point
Intercept calls to execve() to track process creations

Interception happens before new address space created

Modify environment variables passed to child process

A few interesting env. vars we can manipulate from VMM
  - LD_PRELOAD
  - LD_BIND_NOW
  - LD_LIBRARY_PATH

We use LD_PRELOAD in our examples
CODE INJECTION

- Allows VMM to run arbitrary code in guest without cooperation

- Core tool for guest-context virtual services

- Userspace injection: map trusted code into process addr. space

- Kernel: use userspace injection to inject kernel module in guest

- Code can be called directly by VMM, or redirect function calls by patching binary
Process Address Space

mmap area

sizeof(.text) + sizeof(.data)
RWX

Inject .text

Inject .data
Kernel Code Injection

Process Address Space

Inject .text

- `f = open("Inject.ko");`
- `write(f, kobuf);`
- `system("insmod inject.ko");`

Inject .data

- `char * kobuf = "static int mod_enter;
  static int mod_exit;
  ...
  ";`

mmap area

- `sizeof(.text) + sizeof(.data)`
- RWX

.text

.data
### getpid() system call

#### Legacy System Call (INT 0x80)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Latency (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest</td>
<td>4.83</td>
</tr>
<tr>
<td>Guest + intercept</td>
<td>10.24</td>
</tr>
</tbody>
</table>

#### SYSCALL Instruction

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Latency (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest</td>
<td>4.26</td>
</tr>
<tr>
<td>Guest + intercept</td>
<td>4.51</td>
</tr>
</tbody>
</table>

Setup: AMD x86_64, 2.3 GHz Quad-core Opterons  
Host: Fedora 15, Linux 2.6.42  
Guest: Linux 2.6.38
SYSCALL BANDWIDTH UNCHANGED W/ INTERCEPT

![Graph showing syscall bandwidth with and without intercept for different buffer lengths. The x-axis represents buffer length in bytes, ranging from 2B to 2MB. The y-axis represents bandwidth in MB/s, ranging from 0 to 5000 MB/s. Two lines are plotted: one for 'guest' and another for 'guest+intercept'. The graph indicates that the bandwidth with intercept increases significantly compared to the guest only case.]
BANDWIDTH RATIO WITH/WITHOUT INTERCEPT

![Graph showing bandwidth ratio with intercepts and buffer lengths]
Overview
Motivation
GEARS
Evaluation of Tools
Example Service
Conclusions
MPI ACCELERATOR

- MPI library in guest is oblivious to VMs on same host
- Use GEARS to transform MPI_Send/Recv (library calls) into memcpy operations
- Building within VMM is difficult because we lose MPI semantics
  - Discern semantics from guest app
- Uses userspace code injection and process environment modification
MPI ACCELERATOR

[Diagram showing the structure of an MPI application with layers and interactions between the Top Half and Bottom Half.]
MPI ACCELERATOR

- We focus on blocking send and recv

- Injected library redirects some MPI calls as VMM hypercalls

- Bottom half tracks MPI processes using a tuple (VM ID, virtual core, CR3, executable name)
## SERVICE IMPLEMENTATION COMPLEXITY LOW

### MPI Accelerator

<table>
<thead>
<tr>
<th>Component</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preload Wrapper (Top Half)</td>
<td>345</td>
</tr>
<tr>
<td>Kernel Module (Bottom Half)</td>
<td>676</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1021</strong></td>
</tr>
</tbody>
</table>

### Overlay Accelerator

<table>
<thead>
<tr>
<th>Component</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vnet-virtio kernel module (Top Half)</td>
<td>329</td>
</tr>
<tr>
<td>Vnet bridge (Bottom Half)</td>
<td>150</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>479</strong></td>
</tr>
</tbody>
</table>
Overview
Motivation
GEARS
Tools Evaluation
Example Service
Conclusions
CONCLUSIONS

- GEARS, a set of tools to enable guest context virtual services
- Tools that comprise GEARS are few and compact, could be implemented in other VMs
- Developers, with little knowledge of VMM core, and without modifying guest, can use GEARS to build virtual services that are
  - smaller
  - faster
  - easier to understand
  - otherwise unfeasible
Explore boundaries between VMM-injected code and guest code

Safely run trusted components in guest, give them privileged HW access

Application-specific VMM awareness, guest context virtual services as an alternative to OS ABI
Questions?

- Get Palacios (with GEARS) online
  http://v3vee.org/palacios
- Kyle Hale: http://users.eecs.northwestern.edu/~kch479
Latency in Microseconds

Message Size in Bytes

- VNET/P+Gears
- VNET/P
- MPI-Accel
OVERLAY ACCELERATOR

- VNET/P: overlay networking system in Palacios
  - Layer 2 abstraction
  - Near native performance in 1Gbps/10Gbps
  - 75% native throughput, 3-5x native latency fully encapsulated

- Overheads due to VM exits, data copies, data ownership xfer

- Use GEARS to move part of datapath into guest
## VNET ACCELERATOR: SMALL LATENCY IMPROVEMENT

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Native</th>
<th>VNET/P</th>
<th>VNET/P Accel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>0.082 ms</td>
<td>0.255 ms</td>
<td>0.205 ms</td>
</tr>
<tr>
<td>avg</td>
<td>0.204 ms</td>
<td>0.475 ms</td>
<td>0.459 ms</td>
</tr>
<tr>
<td>max</td>
<td>0.403 ms</td>
<td>2.787 ms</td>
<td>2.571 ms</td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td>922 Mbps</td>
<td>901 Mbps</td>
<td>905 Mbps</td>
</tr>
<tr>
<td>TCP</td>
<td>920 Mbps</td>
<td>890 Mbps</td>
<td>898 Mbps</td>
</tr>
</tbody>
</table>

- Proof of concept
- Could improve further with more functionality in guest (with privileged HW access)
WHY CODE INJECTION?

- i.e., why do we care whether a guest cooperates?
- What about a compromised guest? (VMM could forcefully repair a guest)
- What about guests where you don’t have control, but want to enforce some invariant?
- What if changes need to be made on the fly? E.g. host-guest file copy=>saved time
We’ve essentially increased the possible number of attack vectors into the hypervisor, right?

True, but there may be ways we can protect guest-context VMM code better than other interfaces (e.g. Secure in-VM monitoring)

VMM can remove its code from the guest, lock down the guest etc. when a vulnerability is found.