Guest-context Virtual Services

We argue that the implementation of VMM-based services for a guest should extend into the guest itself. VMM-based services are those that a VMM transparently provides to its guests. Examples include virtual devices, isolation between guests, and virtual overlay networking. A guest-context virtual service, however, extends the implementations of these services such that they can span the VMM, guest kernel, and guest application. Guest-context virtual services run directly in the context of the guest OS or application even without its cooperation. They can be simpler than traditional VMM services, they can improve performance, and they can allow new kinds of services that would otherwise not be possible.

System Call Interception

Within the GEARS framework, system calls allow the VMM to keep track of guest activity at a fine granularity. We use various techniques to intercept system calls with low overhead from VMM exits. Our most flexible technique, Fast-selective System Call Exiting, allows the VMM to track user-space events with a mere 5% latency overhead, as shown below. The first table shown is the latency overhead introduced with system call interception for getpid() (the second shows the overhead from a bandwidth perspective for write()).

Environment Modification

In the case of Linux, execve() can be handled as a special case. When this system call is executed, it can be intercepted before a new process environment is created. This allows the VMM to manipulate the new process stack to observe and modify environment variables. Some notable environment variables on Linux for affecting the guest execution path:

- LD_PRELOAD
- LD_BIND_NOW
- LD_LIBRARY_PATH

GEARS

To illustrate the benefits of guest-context virtual services, we developed Guest Examination and Revision Services (GEARS), a framework within the Palacios VMM [1, 2, 3]. This framework allows developers to create guest-context services with minimal familiarity of core VMM internals.

Tools

We claim that there are three tools that are necessary and sufficient to enable a broad range of interesting and useful guest-context services. They are: system call interception, code injection, and process environment modification. These are the three tools that comprise GEARS, and they could be added to other VMs as well. Our implementation shows that each of these tools can be added to a VMM with fairly little development effort, as shown below.

Guest-Context Service Creation

GEARS uses a tiered approach (on both the host and VMM) to inject and run services in a guest. Users provide a code split into two parts, the top-half and the bottom-half. The top-half will run in the guest and the bottom-half, if present, will reside in the VMM. The top and bottom-halves can invoke each other freely, creating a two-way calling interface between the guest and VMM.

The code for the top-half is specific to a particular guest, but we provide host-resident utilities that transform standard code into the guest-specific format. A service developer need only be able to write a normal program or kernel module for the guest in question. If the service requires assistance from the VMM core, the developer can add a bottom half in the form of a host kernel module.

References


Acknowledgements

This project is made possible by support from the United States National Science Foundation (NSF) grant CNS-0709168 and the Department of Energy (DOE) via grant DE-SC0005343.

Code Injection

Code injection is the core tool of the GEARS framework. It allows the VMM to run arbitrary code in guest context without guest cooperation. It relies on the ability to intercept system calls. There are two types of injection, user-space and kernel-space.

User-space:
- Intercept system call in user-space
- mmap() space in heap for inject code
- Copy code pages into process address space
- Invoke code using various methods (e.g. direct setting of RIP, PLT modification)

Kernel-space:
- Load guest-specific kernel module into VMM
- Inject user-space code (as in above) that writes kernel module into guest filesystem

Motivation: MPI library is oblivious to VMs on same physical host.

GEARS Solution: Transform MPI Send/Recv calls into memcpys operations. This isn’t possible from the VMM as MPI semantics are lost, but with GEARS we can recover the semantics from the guest app and achieve the memory bandwidth of the machine for large message sizes. This example service uses environment modification and code injection to redirect MPI library calls.