EXTERIOR: Using a Dual-VM Based External Shell for Guest-OS Introspection, Configuration, and Recovery

ACM VEE ’13
Problem

System administration tasks on a VM from the outside, e.g., issue administrative commands such as `hostname` and `rmmod`.

- One step ahead traditional read-only introspection.
- Benefits.
  - In-box administration capability with out-of-box isolation security guarantee.
  - No need to have user account on the VM, e.g., for cloud service management.
Challenge & solution overview

Challenge: Semantic gap.

- To hypervisor: Raw memory bytes and CPU context.
- To VM: files, APIs, system calls.

Solution overview.

- Dual-VM approach, like process out-grafting: Running a trusted Secure VM (SVM) that has the same kernel with the monitored Guest VM (GVM).
- In other words, maintain a secure duplication of the running VM.
- Redirect memory read/write operations from SVM to GVM.

Approaches that do not fully work.

- Memory diff.
  - Idea: Compute diff via SVM, and apply to GVM.
  - Why no: Only works if data is in kernel data area. Dynamic diff between SVM and GVM (e.g., malware not in SVM).

- Process implanting.
  - Replace a victim and monitor.
  - Why no: Require support library/kernel module in GVM that can be compromised; disrupt victim operations.
Solution

Idea.

- Based on VM space traveler (VMST) work.
- Kernel syscall context aware.
  - Monitor instruction execution of trusted utilities at SVM.
  - Transpareently redirects each individual piece of memory update $M_\delta$ at binary code instruction level from SVM to GVM...
    - ...when syscall of interest gets executed.
- State introspection (e.g., `ps`, `lsmod`, `netstat`) or configuration/recovery tool (e.g., `kill`, `rmmod`).
- Out-of-box program components.
  - SVM: Code, kernel stack (runtime context).
  - GVM: kernel data and heap.

Scope and assumption.

- X86 + Linux.
- Focus on memory state update. Do not deal with external storage.
- Fully trusted and controlled SVM.
- GVM kernel known (through voluntary report or kernel identification).
- No kernel address space layout randomization (ASLR).
- SVM: QEMU for monitoring binary execution.
- GVM: on any hypervisor.
Techniques
Kernel syscall context identification

- Identify process: CR3 + process name (instead of task_struct, use syscall parameter) + esp (lower 12 bits cleared for page alignment).

- System call.
  - Synchronous: int 0x80/iret (Linux specific) or sysenter/sysexit (used since kernel v2.5); system call number in eax.
  - Asynchronous: interrupts and exceptions (software interrupt); spin locks and mutexes are accessed in this context; must be excluded to avoid deadlock.
    - That is why has to use emulator QEMU; such event can be controlled.
    - Trap the entry; leave by iret.
    - More details in VMST.

- Output: Execution context of syscalls, excluding any other kernel execution such as context switch and interrupt/exception handler.
### Techniques

**Kernel syscall context identification**

Not all syscalls are relevant.

```
1  execve("/sbin/sysctl","sysctl","-w","kernel.=1"),...) = 0
2    brk(0)           = 0x604000
3  access("/etc/ld.so.nohwcap",F_OK) = -1 ENSNOT
4  mmap(NULL, 8192, PROT_READ|..., -1, 0) = 0x7f07b1749000
5  access("/etc/ld.so.preload",R_OK) = -1 ENSNOT
6  open("/etc/ld.so.cache", O_RDONLY) = 3
...
47  open("/proc/sys/kernel/randomize_va_space",O_WRONLY|...) = 3
48  fstat(3, [st_mode=S_IFREG|0644, st_size=0, ...]) = 0
49  mmap(NULL, 4096, PROT_READ|..., -1, 0) = 0x7f07b1748000
50  write(3, "l\n", 2) = 2
51  close(3) = 0
...
57  exit_group(0) = ?
```

**Figure 2.** Syscall trace of running `sysctl -w` to turn on the address space randomization in Linux kernel 2.6.32.
Techniques
Kernel data identification and redirection.

- Only kernel global data and heap relevant.
- Without ASLR, global data range is fixed; so trivial.
- To identify heap, first identify stack and minus stack (from esp).
  - Not so straightforward: Stack and heap are not so clear-cut in address space.
  - Stack data dependence tracking algorithm from VMST.
    - Taint analysis: Any data derived from esp and their propagation will be tainted by instrumenting data movement and data arithmetic instructions.
    - For a given kernel address x, if its taint bit is set, then it belongs to kernel stack; otherwise, it is in heap.
- Enumerating syscall of interest: Application specific. Three categories: inspection (e.g., open, read, fstat), configuration (write, sysctl, ioctl, socket), and recovery (e.g., kill, delete_module).
- Identify sync prim in syscalls (e.g., spin_lock/spin_unlock in delete_module).
  - Needs more systematic way to identify than remember address.
  - Instead, forward scanning instruction sequence at function entrance and compare with spin_lock/unlock, _up/_down).
Techniques
Kernel data identification and redirection.

Figure 3. Disassembled Instruction Sequence of spin_lock primitive in different Linux Kernels.
Techniques
Map to GVM memory address.

Figure 4. The VMM Extension of our SVM to Map and Resolve the Physical Memory Address of GVM.

Pause GVM, attach its memory to SVM, and updates.
My thoughts

- This paper, along with its predecessor VMST and process out-grafting, have demonstrated the potential of out-of-box techniques, identified a few tricky places, and suggest some solutions.

- SVM in current implementation is heavily instrumented (through QEMU). Smooth shell will have obvious use (two co-located VM like in process out-grafting).

- The opportunity provided by hardware virtualization: Leverage VM-x (CPU virtualization) and VM-d (IOMMU).

- Work out a hypervisor (KVM/QEMU) extension, allow the host machine to do these fine-grained system administration.

- User interface: On the host machine, the administrator can launch a privileged shell targeted at specific GVM and do the management.
  - Key issue: not trusting the GVM to do the work.
  - While still can do it correctly.