Advanced Topics in Computer Systems, CS262A

Prof. Eric Brewer

Live Migration of Virtual Machines
ReVirt: Virtual-Machine Logging and Replay

Goal of this lecture: illustrate some of value of leveraging the VMM interface for new properties; we cover two here (migration and exact replay), but there are many others as well including debugging and reliability.

I. Live Migration

Migration is useful:

- o Load balancing for long-lived jobs (why not short lived?)
- o Ease of management: controlled maintenance windows
- o Fault tolerance: move job away from flaky (but not yet broken hardware)
- o Energy efficiency: rearrange loads to reduce A/C needs
- o Data center is the right target

Two basic strategies:

- o Local names: move the state physically to the new machine
  - Local memory, CPU registers, local disk (if used -- typically not in data centers)
  - Not really possible for some physical devices, e.g. tape drive
- o Global names: can just use the same name in the new location
  - Network attached storage provides global names for persistent state
  - Network address translation or layer 2 names allows movement of IP addresses

Historically, migration focused on processes:

- o Typically move the process and leave some support for it back on the original machine
  - e.g. old host handles local disk access, forwards network traffic
  - these are "residual dependencies" -- old host must remain up and in use
  - Hard to move exactly the right data for a process -- which bits of the OS must move?
  - E.g. hard to move TCP state of an active connection for a process
- o See Zap paper for best of process-based migration

VMM Migration:

- o Move the whole OS as a unit -- don't need to understand the OS or its state
- o Can move apps for which you have no source code (and are not trusted by the owner)
- o Can avoid residual dependencies in data center thanks to global names
- o Non-live VMM migration is also useful:

- migrate your work environment home and back: put the suspended VMM on a USB key or send it over the network
- Collective project, “Internet suspend and resume”

Goals:

- o Minimize downtime (maximize availability)
- o Keep the total migration time manageable
- o Limit the impact of migration on both the migratee and the local network

Live migration approach:

- o Allocate resources at the destination (to ensure it can receive the domain)
- o Iteratively copy memory pages to the destination host
  - Service continues to run at this time on the source host
  - Any page that gets written will have to be moved again
  - Iterate until a) only small amount remains, or b) not making much forward progress
  - Can increase bandwidth used for later iterations to reduce the time during which pages are dirtied
- o Stop and copy the remaining (dirty) state
  - Service is down during this interval
  - At end of the copy, the source and destination domains are identical and either one could be restarted
  - Once copy is acknowledged, the migration is *committed* in the transactional sense
- o Update IP address to MAC address translation using “gratuitous ARP” packet
  - Service packets starting coming to the new host
  - May lose some packets, but this could have happened anyway and TCP will recover
- o Restart service on the new host
- o Delete domain from the source host (no residual dependencies)

Types of live migration:

- o Managed migration: move the OS without its participation
- o Managed migration with some paravirtualization
  - Stun rogue processes that dirty memory too quickly
  - Move unused pages out of the domain so they don’t need to be copied
- o Self migration: OS participates in the migration (paravirtualization)
  - harder to get a consistent OS snapshot since the OS is running!

Excellent results on all three goals:

- o downtimes are very short (60ms for Quake 3 !)
- o impact on service and network are limited and reasonable
- o total migration time is minutes
- o Once migration is complete, source domain is completely free

Virtual Machines 2

II. ReVirt

Idea: use VM interface to replay non-deterministic attacks exactly

The overall problem:

- Many ways to take over a machine; even the kernel has many bugs
- The number of ways and sophistication is increasing (CERT advisories)
- The PC is too complex to be correct
- So, can't really prevent problems, but we can eliminate holes as we find them
- Goal: make it much easier to find exploited hole after an attack

Basic approach: log the events that took place (audit trail) so that we can reconstruct the attack

- Problem 1: integrity: attacker can change/remove the logs
- Problem 2: the logs are incomplete and may miss key events
- Problem 3: the events may not be enough to recreate non-deterministic bugs; can't decrypt past traffic either since keys are one thing that is non-deterministic

ReVirt approach:

- 1) Log in the VMM to ensure integrity
    - Compromised OS does not have access to VMM logs, even if logger runs in another domain (rather than in the VMM proper)
    - Small code base reduces bugs in VMM
- 2) record non-deterministic events using logs and checkpoints and replay everything **exactly** to pinpoint the exploit
- Narrow VMM interface makes it possible to log all events well -- no need to understand drivers, hardware, etc.

Alternative solutions:

-

OS on OS:

- User mode linux: run linux OS as an "app" inside the real OS
- This is a kind of paravirtualization
- Must map all low-level events to Unix signals
- Use host OS devices instead of raw devices
- ReVirt would work better on top of Xen (which is newer), which is both faster and has a much smaller "trusted computing base"

General replay approach:

- start from a safe checkpoint, then roll forward using the log, watching for the exploit
- May not find it on the first pass, but should learn something. So restart and roll forward again based on new information; repeat. In theory, could implement a "step backward" debugging option, which would restore the checkpoint and roll forward *almost* all the

way to the present (except for the last “step”)

- Insight: only need to log non-deterministic events and inputs, the rest by definition are deterministic and will replay naturally
  - Time: must record the exact time of each event and replay it at the same (virtual) time. For example, replay an interrupt during the exact same *instruction* as before.
  - Input (keyboard, mouse, network packet, etc.): must log the exact input as well as its time. This is easy except for packets, which can consume much space.

ReVirt logging specifics:

- Copy disk image as first checkpoint
- Log events in the VMM into a circular buffer, then periodically move to disk
- For events, log the instruction and the # of branches since the last interrupt; together these define the exact time of the event
- For input, log data from virtual devices: disk, network, real-time clock, keyboard, mouse, CD-ROM, floppy. May not need to log the actual data, if you expect it to be around for replay (e.g. disk blocks)
- For non-deterministic x86 instructions, can trap and emulate to ensure deterministic results (e.g. read cycle counter). (ReVirt actually used paravirtualization instead of emulation.)

ReVirt playback specifics:

- 1) Prevent new events from disturbing playback
- 2) Load checkpoint
- 3) Deliver each event at the precise time. For interrupts: set branch counter to trap every 128 branches until you get close to the right time, then set breakpoint on specific instruction. On each break, check the branch counter to see if this is the right one. If so, issue interrupt. (Breakpoints are slower, so only use them when you get close.)
- Most of playback is near real time.
- Can also replay on a different machine! (similar to live migration)

Tools:

- Continuing from replay: tool can start live from the middle of the replay. This is useful for examining the state of the machine using traditional tools
- Offline tools examine the state of the machine while it is paused. These tools do NOT depend on the OS being correct (unlike first case), and do not interfere with further replay
- X proxy allows replay of the screen

Runtime overhead is fine, typically 1-60% with UM Linux and would be lower with Xen

Replaying complex programs does in fact produce the same results

Final thought: a VMM is also a very effective place for an attacker to reside... install a VMM under the OS and then take over at will, but remain largely undetectable.