Virtual Machine Boot-up Analysis

Abderrahmane Benbachir

Dec 12, 2016

École Polytechnique de Montréal
Laboratoire DORSAL
Agenda

Introduction
Research objectives
Investigation
Results
Future work
Introduction

Boot-up process

- BIOS
  - Basic I/O

- Master Boot Record
  - Execute GRUB

- Boot loader
  - Execute Kernel

- Kernel
  - Execute /sbin/init, initramfs root

- Init process
  - Start run level

- Run level
  - Executed from /etc/rc.d/rc*.d

Still a dark place

Ftrace partially handle this part

LTTng, Ftrace, Dtrace, ...

POLYTECHNIQUE MONTREAL – Abderrahmane Benbachir
Research objectives

• Analyze virtual machines early boot-up

• Detect boot up issues
Investigation

We already know how to trace: **LTTng, Ftrace, Dtrace, Perf.**

Where & How to store traces during bootup?
Investigation

Where & How to store traces during boot up?

Keep them inside Guest vs. Offload them to the Host

- Recording Timestamp
- Synchronization
- Buffer size
- Memory or Disk

- Guest or Host Timestamp?
- Synchronization
- Transfer Channels (Network, Shared Memory, Paravirt)
- Transfer Speed $$$
Investigation

Where & How to store traces during boot up?

Keep them inside Guest vs Offload them to the Host

Light weight Tracers

Network: Sockets

Shared Memory: Qemu Hypertrace

Paravirtualization API: Hypercalls

Must be explored
Qemu Hypertrace

VM#1

Kernel & User space

Control

Notify

Config

Write

Data channel

Wake up

Hypertrace Softmmu

Read

Emit User space events

Host Kernel

KVM.KO

By Lluís Vilanova

Shared Memory Btw Guest & Qemu
Hypercall

Direct Execution of Guest Requests

This layer is not involved

VM#1

User space

Paravirtualized Guest OS

Qemu

Hypercall

Hypercall

Host Kernel

KVM.KO

Hypercall

Hardware
1.99 us: writing to control channel

- Mean: 2070 ns
- Median: 1995 ns
- Std deviation: 1580 ns
Benchmarks

Hypercall Overhead

310 ns: `kvm_exit + kvm_hypercall + kvm_entry`

323 ns

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>340 ns</td>
</tr>
<tr>
<td>Median</td>
<td>310 ns</td>
</tr>
<tr>
<td>Std deviation</td>
<td>833 ns</td>
</tr>
</tbody>
</table>
Hypercalls Implementation

How to trace through Hypercalls

• Hook to Ftrace “function graph” callback for entry & exit
• Only trace the Host
• Use only host Timestamp : No synchronization required
• Dump kernel symbols to map each function names
Hypercalls Implementation

host kernel trace with LTTng

hypercall payload = 5 args x 64 bits = 40 Bytes

hypercall(nr, a0, a1, a2, a3)

[13:59:00.035571626] (+0.000000494) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579284592, a1 = 0, a2 = 0, a3 = 0 }

[13:59:00.035572670] (+0.000000519) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579288208, a1 = 1, a2 = 4198, a3 = 3 }

[13:59:00.035573178] (+0.000000508) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579292656, a1 = 1, a2 = 5227, a3 = 2 }

[13:59:00.035573686] (+0.000000508) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579922416, a1 = 1, a2 = 6251, a3 = 1 }

[13:59:39.016994097] (+0.000000482) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 0, a2 = 0, a3 = 0 }

[13:59:00.035584321] (+0.000000531) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579265552, a1 = 0, a2 = 0, a3 = 0 }

[13:59:00.035586871] (+0.000000490) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579284528, a1 = 0, a2 = 0, a3 = 0 }

[13:59:00.035587383] (+0.000000525) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579284592, a1 = 0, a2 = 0, a3 = 0 }

[13:59:00.035585361] (+0.000000503) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071587317584, a1 = 0, a2 = 0, a3 = 0 }

[13:59:39.016994570] (+0.000000473) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 0, a2 = 0, a3 = 0 }

[13:59:39.016995047] (+0.000000477) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4196150, a1 = 0, a2 = 0, a3 = 0 }

[13:59:39.016996002] (+0.000000481) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4196518, a1 = 0, a2 = 0, a3 = 0 }

[13:59:00.035585361] (+0.000000503) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579284592, a1 = 0, a2 = 0, a3 = 0 }

[13:59:00.035585863] (+0.000000502) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579284592, a1 = 0, a2 = 0, a3 = 0 }

[13:59:39.017128748] (+0.000000734) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4196518, a1 = 0, a2 = 0, a3 = 0 }

[13:59:39.017129250] (+0.000000502) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 0, a2 = 0, a3 = 0 }

[13:59:39.017129783] (+0.000000502) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 0, a2 = 0, a3 = 0 }

[13:59:00.035585361] (+0.000000503) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 1000, a0 = 18446744071579284592, a1 = 0, a2 = 0, a3 = 0 }

[13:59:39.017129250] (+0.000000502) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 0, a2 = 0, a3 = 0 }

[13:59:39.017129783] (+0.000000502) abder-pc kvm_x86_hypercall: { cpu_id = 3 }, { nr = 2000, a0 = 4197127, a1 = 0, a2 = 0, a3 = 0 }
Function graph

Callstacks of Guest Kernel space

300 us
Hybrid tracing technique

Kernel space + User space
Overhead by function (sorted by median)
Function Graph Overhead (Sorted by median)

POLYTECHNIQUE MONTREAL – Abderrahmane Benbachir
Future Work

- Apply hypercall technique to trace full kernel boot up
- Explore this solution with nested virtual machine
- Can we trace the Boot loader too?
Questions

Abderrahmane.benbachir@polymtl.ca

https://github.com/abenbachir

Thanks to Professor Michel Dagenais and our partners EfficiOS and Ericsson.
References

Hypercall Implementation: https://gist.github.com/abenbachir/344822b5ba9fc5ac384cdec3f087e018

QEMU Hypertrace Patches: http://patchwork.ozlabs.org/project/qemu-devel/list/?state=&q=Hypertrace&archive