Verification of a Separation Kernel

Inzemamul Haque

Indian Institute of Science, Bangalore

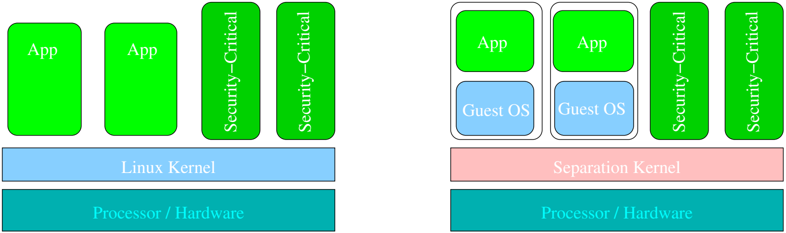
17 July 2017

Outline

Motivation

- Defense and aerospace applications need to run security-critical programs along with untrusted programs, on the same machine.
- Commercial O/Ss have many vulnerabilities which make them unsuitable for this task.
- A Separation Kernel provides such a solution.
- Would like to prove certain security properties of a separation kernel.
- Formal verification gives highest level of assurance that a system satisfies a required property.

Separation Kernel



Objective

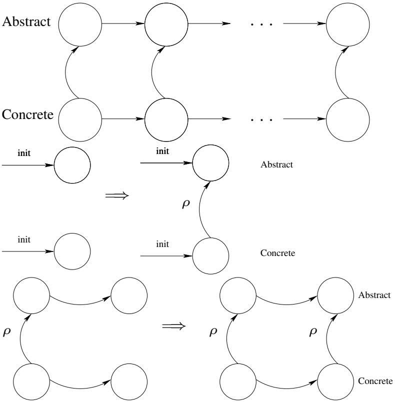
Goal

To give a machine-checked proof of correctness of a separation kernel.

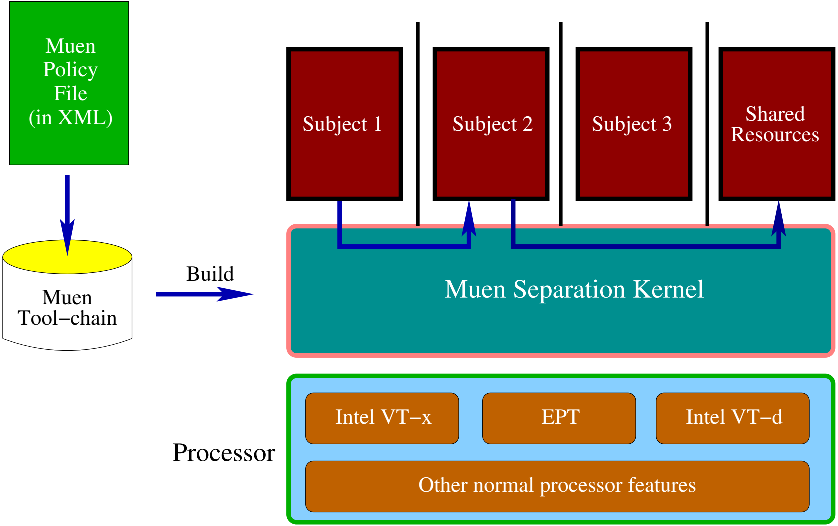
- How does it address the security concern?
- Security is part of the abstract model.

Methodology

- Define an abstract model which captures the correct behaviour of the separation kernel.
- To show that for every execution in the concrete there is a corresponding execution in the abstract.
- Inductive proof by defining an abstraction relation.



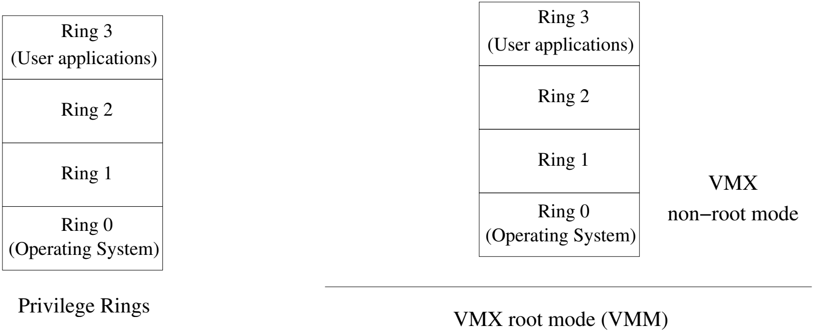
Muen Separation Kernel



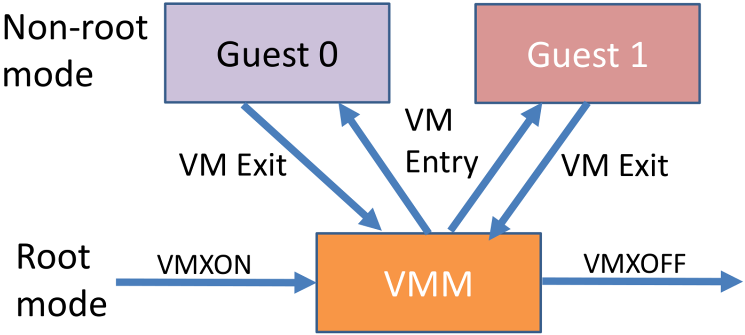
Example Policy File


```
- <system>
  + <features>
    <include href="common_platform.xml"/>
  + <kernelDiagnosticsDevice physical="debugconsole">
  - <memory>
      <include href="common_memory.xml"/>
      <memory caching="WB" alignment="16#0020_0000#" size="16#0f60_0000#" name="nic_linux|ram"/>
      <memory caching="WB" size="16#0008_0000#" name="nic_linux|lowmem"/>
      <memory caching="WB" alignment="16#0020_0000#" size="16#0f60_0000#" name="storage_linux|ram"/>
      <memory caching="WB" size="16#0008_0000#" name="storage_linux|lowmem"/>
      <memory caching="WB" size="16#0001_0000#" name="logbuffer_placeholder0"/>
      <memory caching="WB" size="16#0002_0000#" name="logbuffer_placeholder"/>
    </memory>
    <include href="common_events.xml"/>
    <include href="common_channels.xml"/>
    <include href="common_components.xml"/>
  - <subjects>
      <include href="subject_vt.xml"/>
      <include href="subject_nic_sm.xml"/>
      <include href="subject_storage_sm.xml"/>
    - <subject name="nic_linux" profile="linux">
        <bootparams>console=hvc console=ttyS0 pci=noearly hostname=nic_linux</bootparams>
      - <memory>
          <memory physical="nic_linux|lowmem" executable="false" writable="true" virtualAddress="16#0002_0000#" logical="lowmem"/>
          <memory physical="initramfs" executable="false" writable="false" virtualAddress="16#00a0_0000#" logical="initramfs"/>
          <memory physical="nic_linux|ram" executable="true" writable="true" virtualAddress="16#00f0_0000#" logical="ram"/>
        </memory>
      + <devices>
      + <events>
      - <channels>
          <reader physical="virtual_input_1" virtualAddress="16#3000#" logical="virtual_input" vector="49"/>
          <writer physical="virtual_console_1" virtualAddress="16#4000#" logical="virtual_console" event="1"/>
          <reader physical="testchannel_2" virtualAddress="16#00e0_0000#" logical="testchannel_2"/>
          <writer physical="testchannel_1" virtualAddress="16#00e0_1000#" logical="testchannel_1"/>
          <reader physical="testchannel_4" virtualAddress="16#00e0_2000#" logical="testchannel_4"/>
          <writer physical="testchannel_3" virtualAddress="16#00e0_3000#" logical="testchannel_3"/>
        </channels>
        <component ref="linux"/>
```

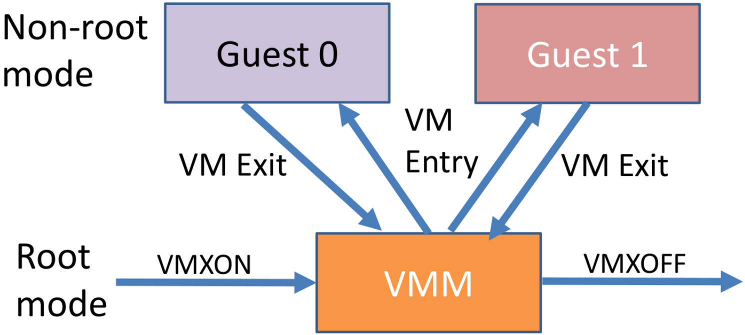
Intel VT-x


Ring 3
(User applications)
Ring 2
Ring 1
Ring 0
(Operating System)
Privilege Rings
Ring 3
(User applications)
Ring 2
Ring 1
Ring 0
(Operating System)
VMX
non–root mode
VMX root mode (VMM)

Life-cycle of a VMM

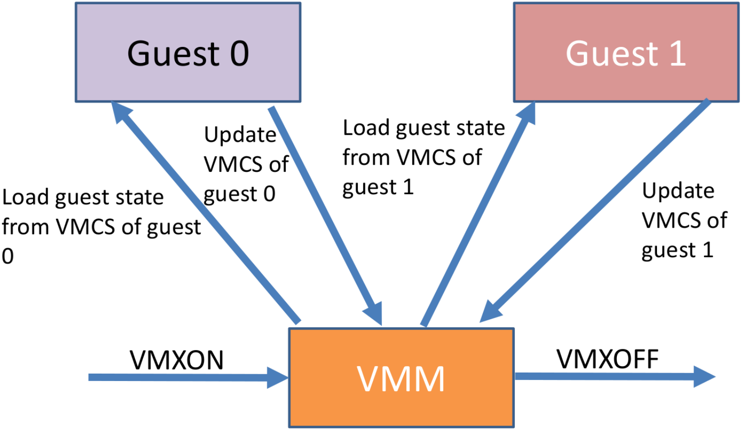


Life-cycle of a VMM



How to manage states during VM-entry and VM-exit?

Virtual Machine Control Structure (VMCS)



VMCS Data

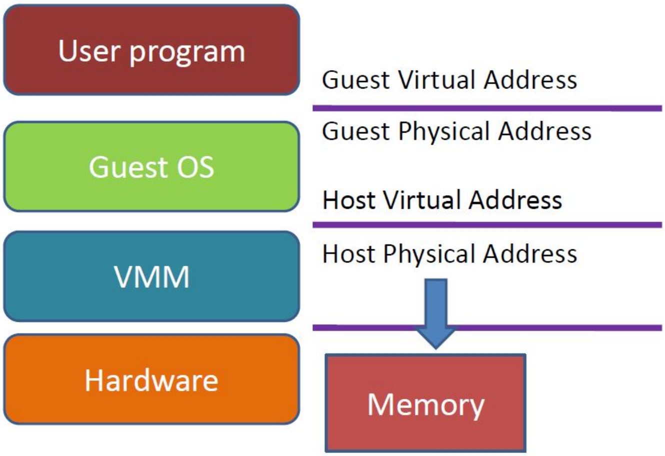
Fields in VMCS can be classified as following:

- Guest state area - mainly register state of the guest
- Host state area - processor state to be loaded at VM exits
- VM-execution control fields - fields like external interrupt exiting, CR3 load exiting, etc.
- VM-entry control fields - fields which tell what to be saved during VM entry.
- VM-exit control fields - fields which tell what to be saved during VM exit.
- VM-exit information fields

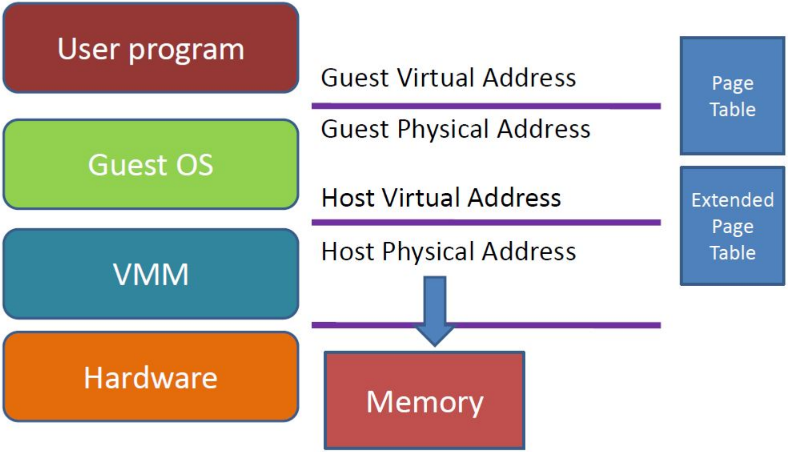
Causes of VM-Exit

- Instructions causing unconditional exits
  - INVD, CPUID, etc.
- Instructions causing conditional exits
  - HLT, if HLT-exiting field is set
  - Mov from CR3, if CR3-exiting field is set
- External interrupts if external interrupt exiting field is set.
- VMX preemption timer counts to zero.

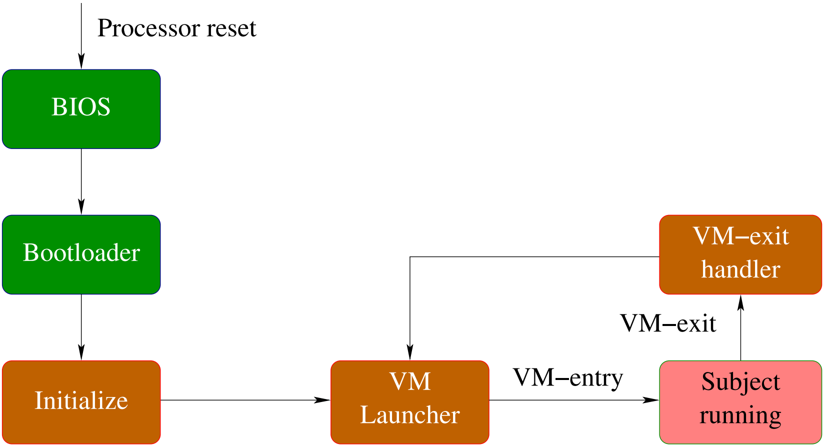
Extended Page Tables



Extended Page Tables



Muen Separation Kernel



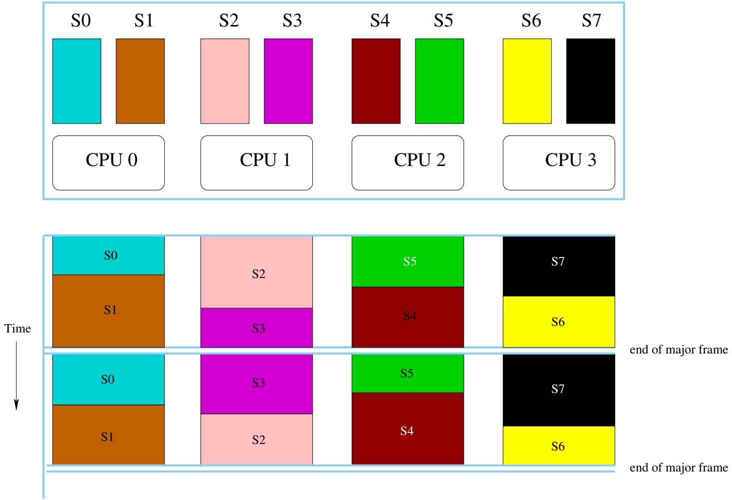
Challenges

- Dealing with the mixture of assembly and Ada.
- Proof for a general policy
- Reasoning about the invariants involved

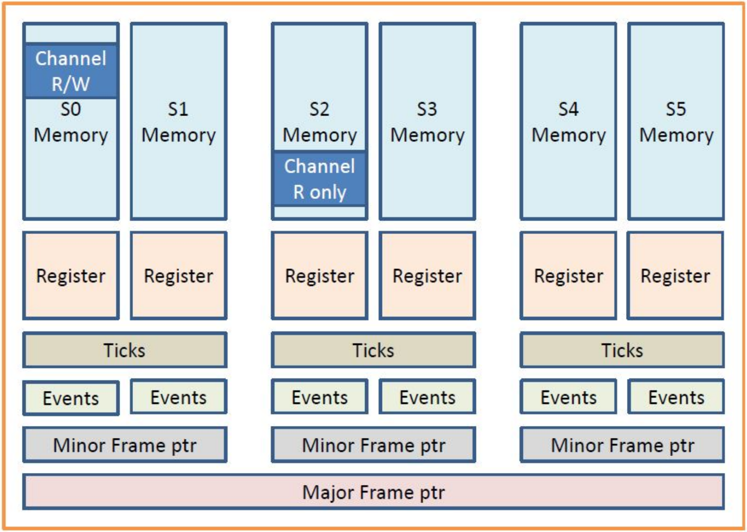
Abstract Model

- Our model is a state transition system.
- Policy also specifies number of CPUs and order of execution of subjects.
- Every subject runs on a standalone machine according to the schedule specified in the policy.

Abstract Model



State in the Model



Transitions in the Model

- Tick
- Local operation - memory accessed by the subjects
- External interrupt
- Events
- Read channel
- Write channel

AdaCore SPARK

- Tool to prove certain properties of Ada programs like
  - satisfiability of pre- and post-conditions for a program.
  - checking assertions at certain points in the program.
  - absence of run-time errors like division by zero, dangling pointers.
- Carried out small exercise to verify virtual memory translator.

```
--Ghost function to check invariants.
function CheckInvariants (loc : in Natural) return Boolean
is
  (
   (loc <Size)
   and
   --Invariant to check alignment
   --(invariant \forall unsigned i,j; (i<256 && j<256) ==> ((pd[i][j])%256==0))
   (for all i in 0 .. 255 =>(for all j in 0 .. 255 => (PD(i,j)mod 256 =0) ))
   and
   --(invariant \forall unsigned i,j; (i<256 && j<256) ==> (pd[i][j] < SIZE))
   (for all i in 0 .. 255 => (for all j in 0.. 255 => PD(i,j)<Size ))
   and
   --Invariant to check that two distinct pages do not map to the same frames
   --(invariant \forall unsigned i,j,p,q; (i<256 && j<256 && p<256 && q<256 && i != p && j != q) ==>
     (for all i in 0.. 255 =>
          (for all j in 0.. 255=>
               (for all p in 0.. 255 =>
                  (for all q in 0.. 255 =>
                     (if (i/=p or j/=q) then
                          (PD(i,j) /= PD(p,q)))))))
   and
   --Gluing invariant
   --Both physical memory and abstract memory contain same data.
   --(invariant \forall unsigned i,hi,mid,lo,page; (i < SIZE && hi == i/65536 && mid == (i%65536)/25
   (for all i in 0.. Size-1 => (
                      (PD(i/65536,(i mod 65536)/256) <= Size-256)))
   and
   (for all i in 0.. Size-1 => (if (PD(i/65536,(i mod 65536)/256) <= Size-256) then
                                 Mem(PD(i/65536,(i mod 65536)/256)+ (i mod 256))=AbsMemory(i)))
   )
  with Ghost;
```

Dealing with mixture of assembly and Ada code

- Writing the assembly instructions as Ada functions.
- e.g. a 64-bit register as a 64-bit modular datatype in Ada

```
package Assembly
is
   type Byte is mod 2**8;
   for Byte'Size use 8;

   type Word16 is mod 2**16;
   for Word16'Size use 16;

   type Word32 is mod 2**32;
   for Word32'Size use 32;

   type Word64 is mod 2**64;
   for Word64'Size use 64;
```

```
type CPU_Registers_Type64 is record
   CR2 : Word64;
   RAX : Word64;
   RBX : Word64;
   RCX : Word64;
   RDX : Word64;
   RDI : Word64;
   RSI : Word64;
   RBP : Word64;
   R08 : Word64;
   R09 : Word64;
   R10 : Word64;
   R11 : Word64;
   R12 : Word64;
   R13 : Word64;
   R14 : Word64;
   R15 : Word64;
   RFLAGS : Word64;
end record;
```

Conclusion

- Giving a machine checked proof of correctness of a separation kernel
- We have modelled the Muen separation kernel
- Focusing on correctness of initialization part of the kernel as of now.
- Initially working on a fixed policy