# Modeling co-tenant risk for cloud services

## (ISC)$^2$ Melbourne Chapter Meeting - 14/07/2021

Michiel Kalkman DigIO / Mantel Group

# Some background

Figure 1: It's turtles all the way down[1]

[1] https://openclipart.org/detail/254346/pyramid-of-turtle

# The plan

- Remain vendor-neutral
- Find the right levels of abstraction
- Have a grounding in technical reality
- Be comprehensible to non-technical stakeholders
- Have a low barrier to entry
- Be portable
- Be repeatable

Section 1

# Communication

# Actors and actions

- Three actors
  - Tenant (Us)
  - Co-Tenant (Them)
  - Provider
- Flow of control
  - Transition of the ability to execute instructions on the shared infrastructure
- Containment
  - Controls placed on an actor to prevent policy violations
- Isolation
  - Degree to which one actor is unaffected by the actions of another
- Exposure
  - Degree to which one actor is affected by the actions of another

# Actor colours

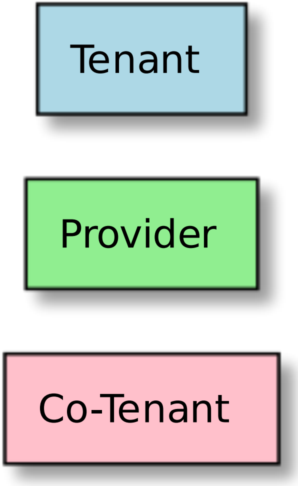


Figure 2: Colour coding

## Colours

- Areas under tenant control are blue (us)
- Areas under co-tenant control are red (untrusted)
- Areas under provider control are green (trusted)

# Actor relationships

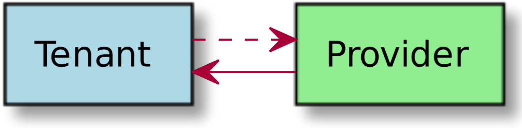


Figure 3: Connections

## Connections

- Connection lines represent **flow of control**
- Dashed or dotted lines are **gated** - these have restrictions (preventative controls) on them.
- Solid lines are **ungated** - these have no restrictions. Some have audit points that can be used to create detective controls

# Area boundaries

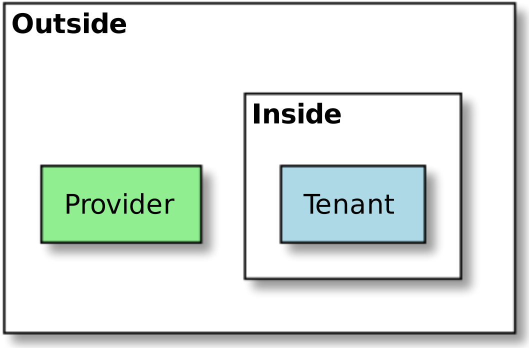


Figure 4: Boundaries

## Boundaries

- Boundaries are containing boxes
- Boxes are nestable
- Outside area contains all boxes within it
    - Outer boxes have more privileges
    - Inner boxes have fewer privileges

# Actor boundaries

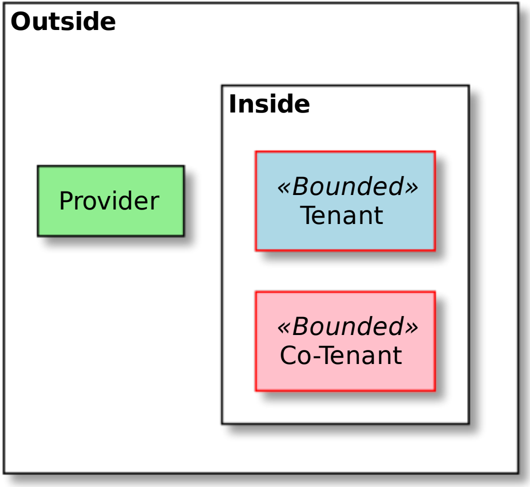


Figure 5: Boundaries

## Boundaries

- Actors with a **red** border have restrictions within the boundary
  - These are also explicity marked as <<Bounded>>
- Actors **without a red border** have unrestricted access within the boundary

Section 2

# Model scenarios

# Single mode CPU

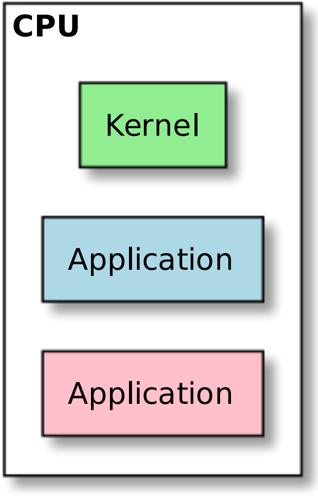


Figure 6: Privilege model - Single mode CPU

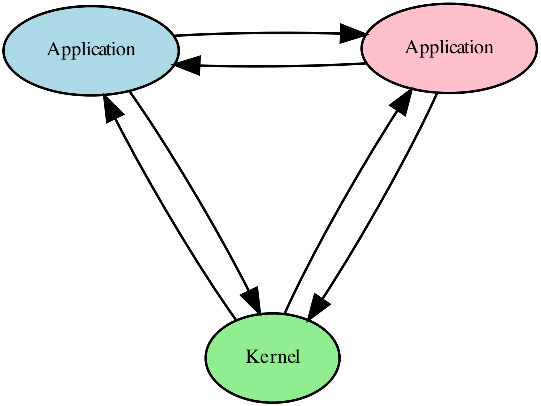


Figure 7: Execution flow - Single mode CPU

# Privilege model Intel IA32/64 - multi-mode CPU

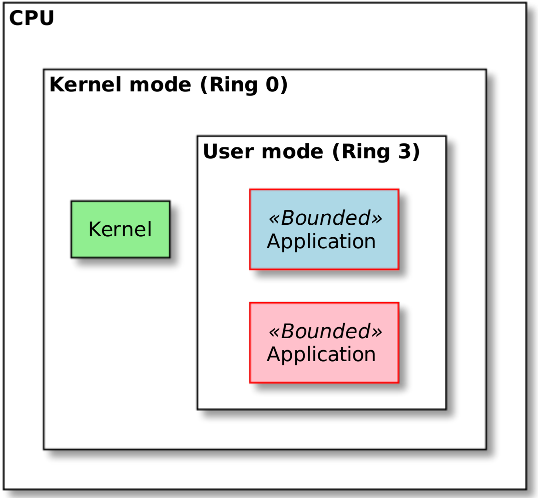


Figure 8: Intel x86 modes (*rings*)

## User mode restrictions

- Cannot run privileged instructions
- Cannot write to all registers
- Cannot modify current segment register (*i.e. change rings*)
- Cannot modify page tables
- Cannot modify CR3 register, this prevents seeing other processes' memory
- Cannot register interrupt handlers
- Cannot use IO instructions (e.g. `in`, `out`)

# Execution flow Intel IA32/64

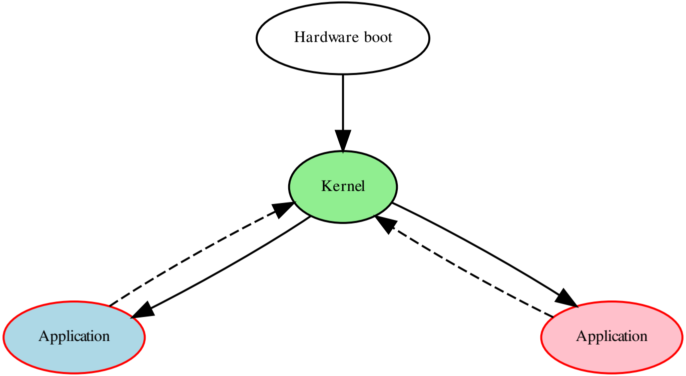


Figure 9: Intel x86 kernel/application flow

Modes are used to isolate processes from each other, the kernel and, by implication, system resources. Only kernel code can, for example, execute IO instructions.

# Virtualization - Definition

- Virtualization constructs isomorphism from guest to host, by implementing functions `V()` and `E()`
- All guest state `S` is mapped onto host state `S'` through a function `V(S)`
- For every state change operation `E(S)` in the guest is a corresponding state change `E'(S')` in the host

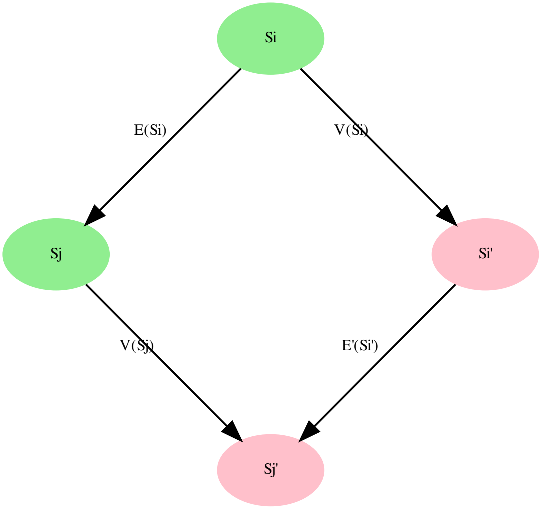


Figure 10: Popek and Goldberg (PG74)

# Virtualization - Privilege model Intel IA32/64

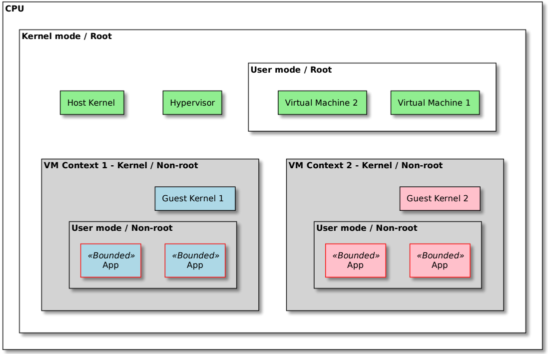


Figure 11: IA32/64 zones (Provider hosts VMs)

## Root and Non-root mode

- Virtual environments contained in VM contexts
- Virtual environments operate in **non-root** mode
- Only **root** mode has access to VMX instructions
- Hypervisor (or VM Monitor) creates and runs VMs
- Computer on which hypervisor runs is called **Host**
- Computer on which VM runs is called **Guest**

# Virtualization - Execution flow Intel IA32/64

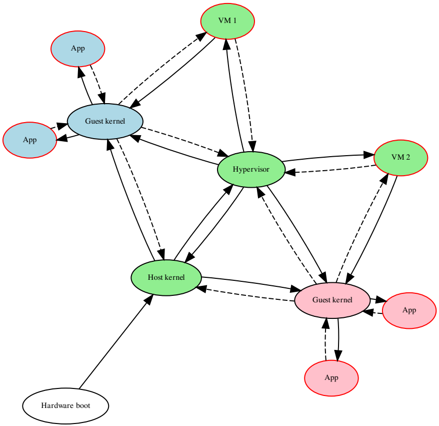


Figure 12: IA32/64 flow (Provider hosts VMs)

## Notes

- Connection between Guest Kernel and VM is mediated by Hypervisor
  - Relevant instructions are trapped by Hypervisor and control is then passed to VM
  - This is a pass-through mechanism, modeling it as a direct connection makes is easier to reason about the attack surface
- All gated connections have boundaries enforced via CPU mechanisms

# Model and flow - Provider hosts VMs

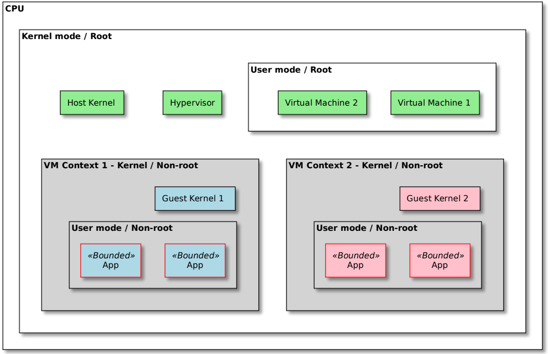


Figure 13: Zones

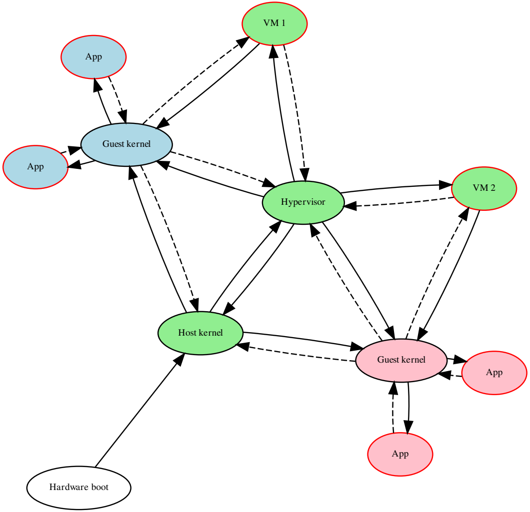


Figure 14: Flow

# Model and flow - Provider hosts applications

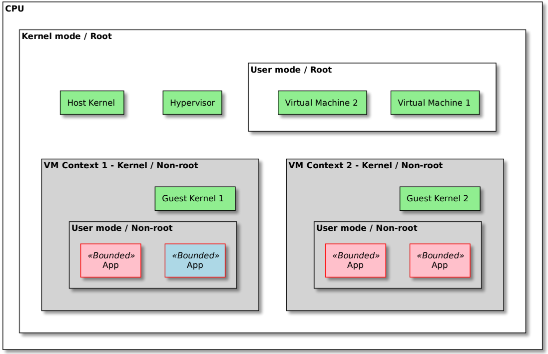


Figure 15: Zones

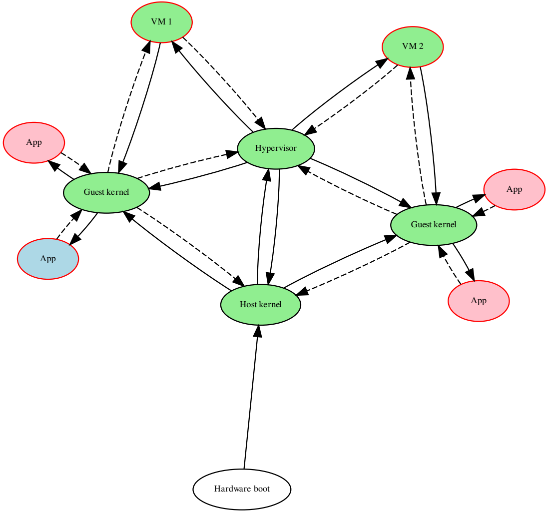


Figure 16: Flow

# Model and flow - Provider hosts nested VMs

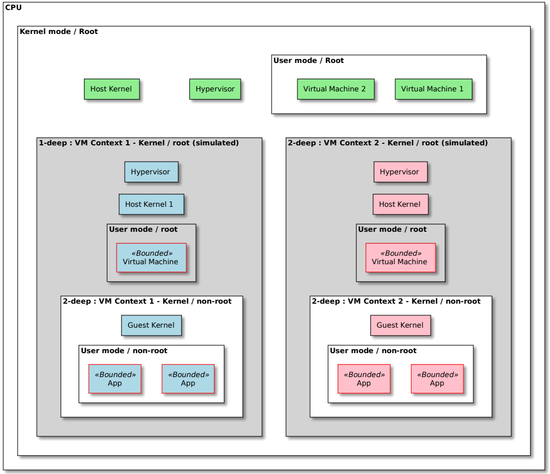


Figure 17: Zones

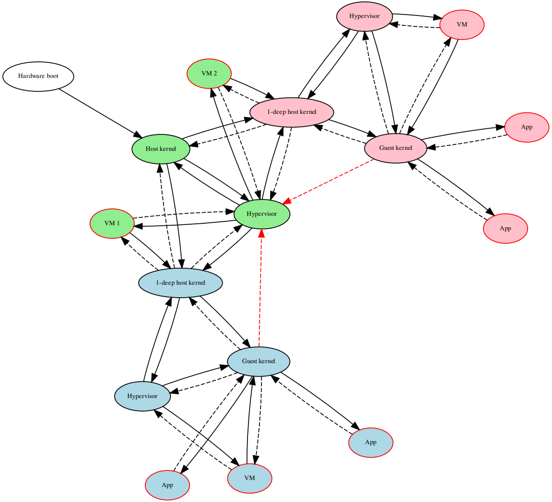


Figure 18: Flow

# Gate types

| Higher privilege | Lower Privilege | Gate type |
|---|---|---|
| Kernel / root | User / root | System call |
| Kernel / root | Kernel / non-root | Virtualization trap |
| Kernel / non-root | User / non-root | System call |
| Kernel / non-root | User / root | Virtualization trap (via VMM) |

# The kernel / user boundary

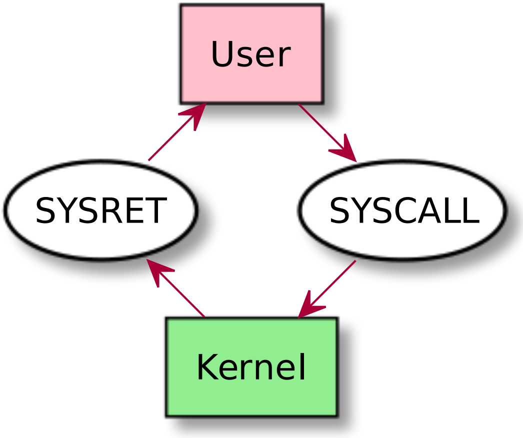


Figure 19: Kernel/User control flow

## Enter Ring 3

- `sysret`, `sysenter`, `iret`

## Exit Ring 3

- `syscall`, `sysenter`
- Software interrupt (Linux 0x80)
- Trap
- Far call

# The host / guest boundary

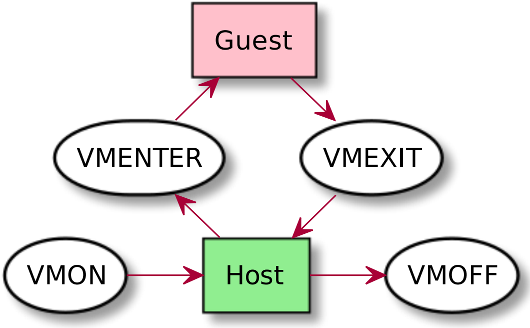


Figure 20: Host/Guest control flow

## VMEXIT scenarios

- Any guest instruction that causes an exception
- An external I/O interrupt
- Root-mode sensitive x86 privileged or sensitive instructions (e.g. `hlt`, `pause`)
- Hypercalls - `vmcall` - Explicit transition from non-root to root
- VT-x ISA extensions to control non-root execution (e.g. `vmclear`, `vmlaunch`)

Section 3

# Qualify scenarios

# Combined controls User/Kernel

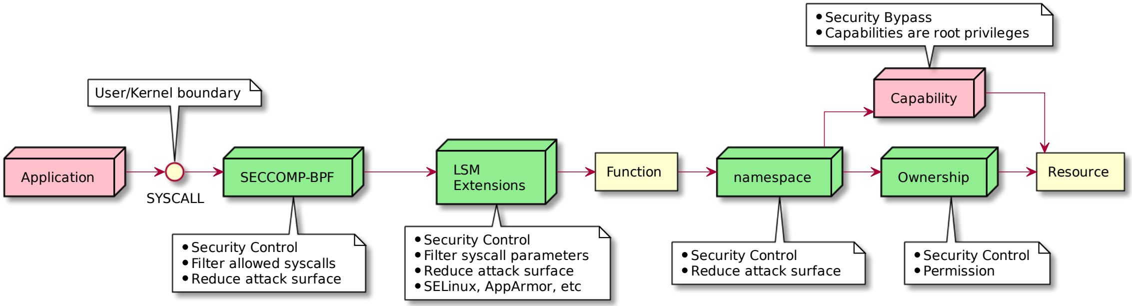


Figure 21: Security controls on the User/Kernel boundary

# Attack Model - User to Kernel

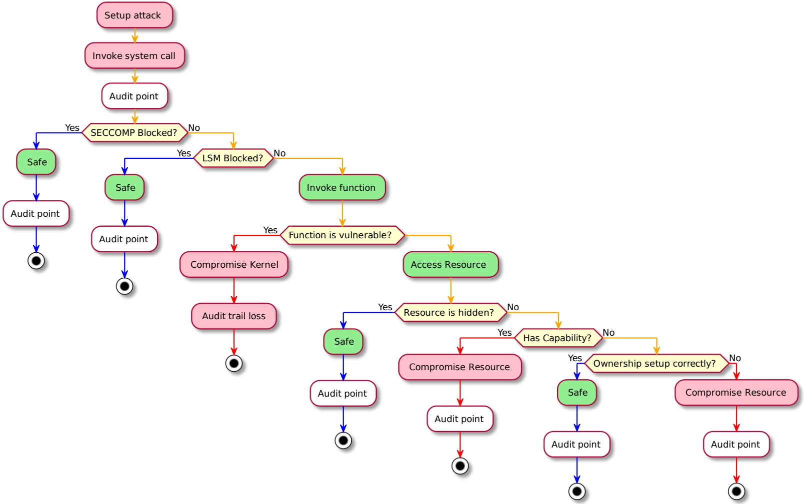


Figure 22: User/Kernel controls

# Section 4

## Quantify scenarios

# Quantifying boundary isolation

- Score the boundary on a scale from 0 to 100
- 0 is completely exposed
- 100 is completely isolated
- List all controls
- Select a maximum number for all controls activated
    - Distribute over available controls
    - Leave the remainder as residual risk

# Residual Risk Example : POP SS / MOV SS vulnerability

**CVE-2018-8897**

> *If the instruction following the MOV to SS or POP to SS instruction is an instruction like SYSCALL, SYSENTER, INT 3, etc. that transfers control to the operating system at CPL $< 3$, the debug exception is delivered after the transfer to CPL $< 3$ is complete. OS kernels may not expect this order of events and may therefore experience unexpected behavior when it occurs.* [2]

**Rough translation**

A handler set in Ring 3 can be called while still in Ring 0

[2] https://nvd.nist.gov/vuln/detail/CVE-2018-8897

# Containing Userspace processes (Linux)

- Max isolation with full controls 75
- Minimum exposure of 25

Breakdown (example scoring for demonstration purposes),

| Aspect | Rating |
|---|---|
| SECCOMP | +30 |
| LSM | +25 |
| Permissions | +10 |
| Namespaces | +10 |
| Capabilities | −60 |

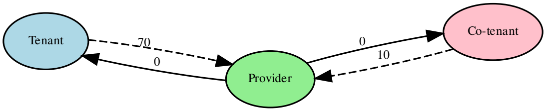


Figure 23: Note this relationship is asymmetric

| SYSCALL controls | Tenant | Co-tenant |
|---|---|---|
| SECCOMP | +30 | +30 |
| LSM | +25 | +25 |
| ACL/Permissions | +15 | +15 |
| Namespaces | | |
| Capabilities | | −60 |
| **Score** | **70** | **10** |

# Containment model hosted virtual machines

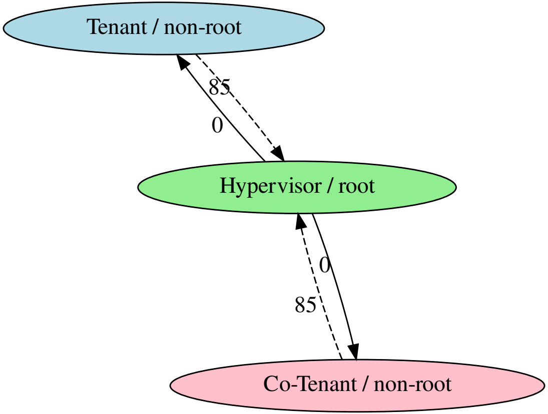


Figure 24: Simplified (VM+VMM as single unit)

| Threat origin | Path |
|---|---|
| Host peer | [85, 0] |

# Residual Risk Example : VENOM

**CVE-2015-3456**

> *The Floppy Disk Controller (FDC) in QEMU, as used in Xen 4.5.x and earlier and KVM, allows local guest users to cause a denial of service (out-of-bounds write and guest crash) or possibly execute arbitrary code via the (1) FD_CMD_READ_ID, (2) FD_CMD_DRIVE_SPECIFICATION_COMMAND, or other unspecified commands, aka VENOM.* [3]

**Rough translation**

Back in 2004 Floppy Disks were still a thing and a driver was added to QEMU. No-one has looked at it since, QEMU is used in various VMMs, it contains the driver and that is vulnerable to a buffer overflow.

[3] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3456

# Containment model hosted applications/functions

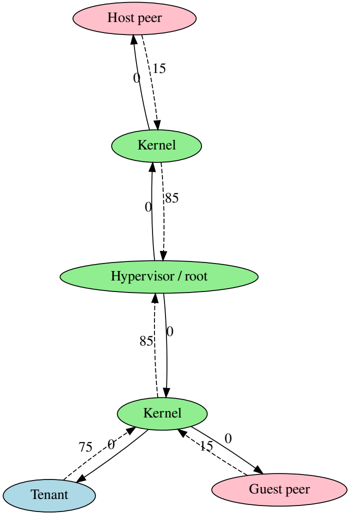


Figure 25: Simplified (VM+VMM as single unit)

| Threat origin | Path |
|---|---|
| Guest peer | [15, 0] |
| Host peer | [15, 85, 0, 0] |

# Containment model JVM

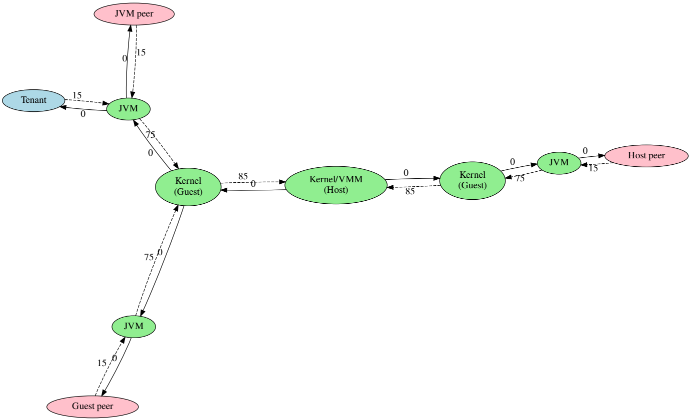


Figure 26: As a weighted, directed graph

| Threat origin | Path |
|---|---|
| JVM peer | [15, 0] |
| Guest peer | [15, 75, 0, 0] |
| Host peer | [15, 75, 85, 0, 0, 0] |

# Exposure analysis

## Calculation

```
function Exposure(arr) {
  return (arr.reduce((b, a)=> {
    return ( 1 - (a/100)) * b;
  }, 1) * 100).toFixed(2);
}
```

| Threat origin | Path | Exposure() |
|---|---|---|
| JVM peer | [15, 0] | 85.00 |
| Guest peer | [15, 75, 0, 0] | 21.25 |
| Host peer | [15, 75, 85, 0, 0, 0] | 3.19 |

# Example - Kubernetes

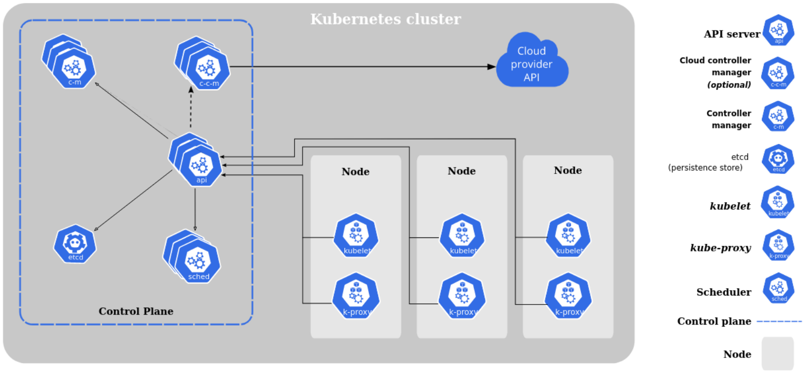


Figure 27: Kubernetes components[4]

[4] https://kubernetes.io/docs/concepts/overview/components/

# Example - Managed Kubernetes

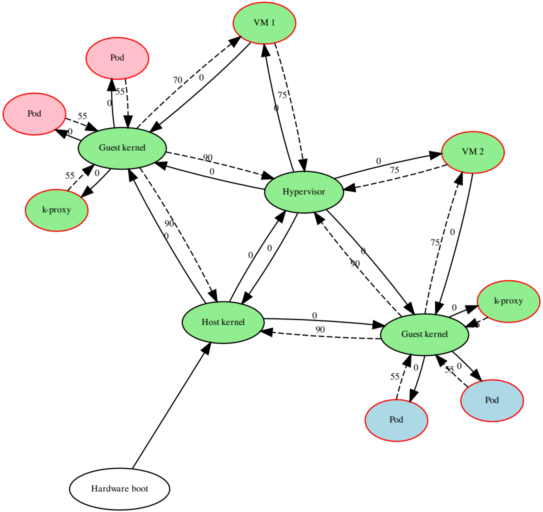


Figure 28: Separate nodes for tenants

| Threat origin | Path | Exposure |
|---|---|---|
| Host peer | [55,90,0,0] | 4.50 |
| Host peer | [55,75,75,0,0] | 2.81 |

# Example - Kubernetes - Shared tenancy

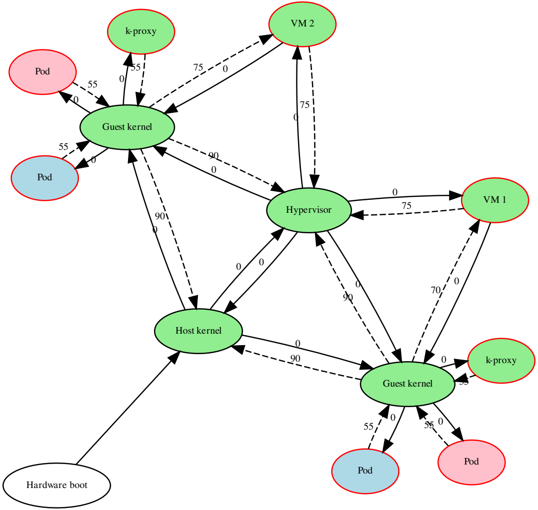


Figure 29: Shared nodes, separate k8s namespaces for tenants

| Threat origin | Path | Exposure |
|---|---|---|
| Guest peer | [55] | 45.00 |
| Host peer | [55,90,0,0] | 4.50 |
| Host peer | [55,75,75,0,0] | 2.81 |

# Example - Firecracker

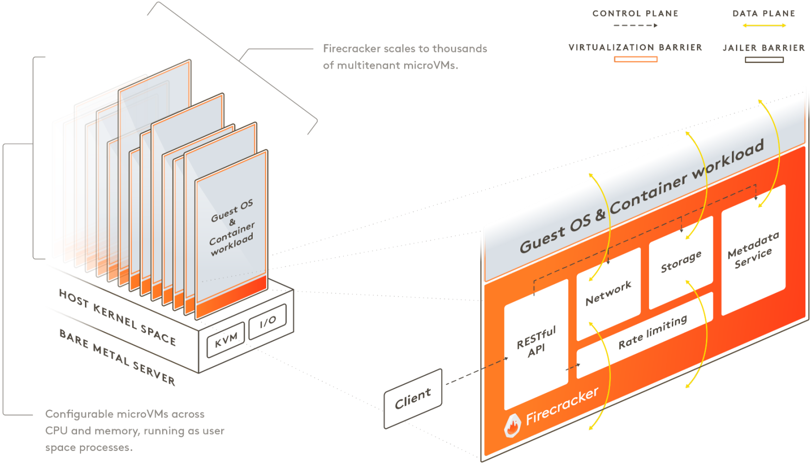


Figure 30: Firecracker components[5]

[5]https://firecracker-microvm.github.io/

# Example - Firecracker model

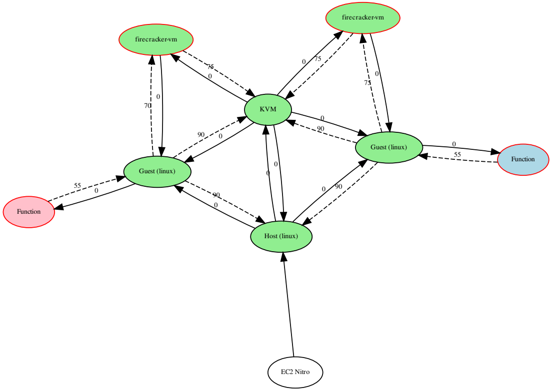


Figure 31: Separate nodes for tenants

| Threat origin | Path | Exposure |
|---|---|---|
| Host peer | [55,95,0,0] | 4.50 |
| Host peer | [55,75,75,0,0] | 2.81 |

- Kernel/User controls on *Function*
- Minimal VM implementation in *Rust*
- Hardened KVM
- Hardened, minimal OS for Guest and Host

Section 5

# Side channels

# L1TF - L1 Terminal Fault

*L1 Terminal Fault is a **hardware vulnerability** which allows unprivileged speculative access to data which is available in the Level 1 Data Cache [..]*

*L1TF allows to attack **any physical memory** address in the system and the attack works **across all protection domains**.*

*The fact that L1TF breaks all domain protections allows malicious guest OSes, which can control the PTEs directly, and malicious guest user space applications, which run on an **unprotected guest kernel** lacking the PTE inversion mitigation for L1TF, to attack physical host memory.* [6]

[6] https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/l1tf.html

# L1TF Containment model

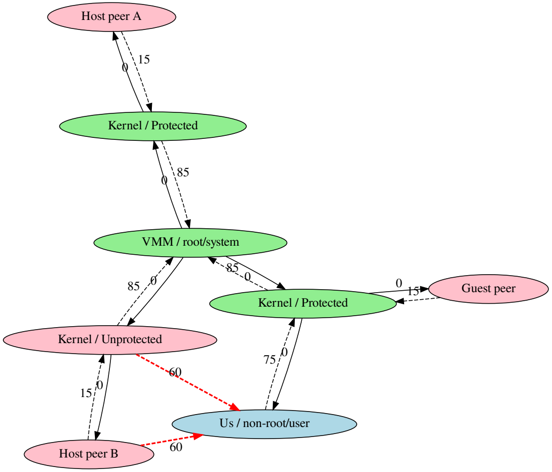


Figure 32: As a weighted, directed graph

| Threat origin | Path |
|---|---|
| Guest peer | [15, 0] |
| Host peer A | [15, 85, 0, 0] |
| Host peer B | [15, 85, 0, 0] |
| Host peer B | [60] |

| Threat origin | Exposure |
|---|---|
| Guest peer | 85.00 |
| Host peer A | 12.75 |
| Host peer B | 12.75 |
| Host peer B | 40.00 |
| B combined | 47.65 |

Section 6

# Further exploration

# Sample vendor technology

| Vendor | Host kernel | Hypervisor | Virtual Machine |
|---|---|---|---|
| GCP[7] | Linux | KVM variant (custom) | In house |
| Azure[8] | Windows | Hyper-V / Azure-V | ? |
| AWS[9] | Linux | KVM | Firecracker-VM |
| DO[10] | Linux | KVM | QEMU |

[7] 7 ways we harden our KVM hypervisor at Google Cloud
[8] Hypervisor security on the Azure fleet
[9] Firecracker – Lightweight Virtualization for Serverless Computing
[10] Open Source at DigitalOcean: Introducing go-qemu and go-libvirt

# Blast radius

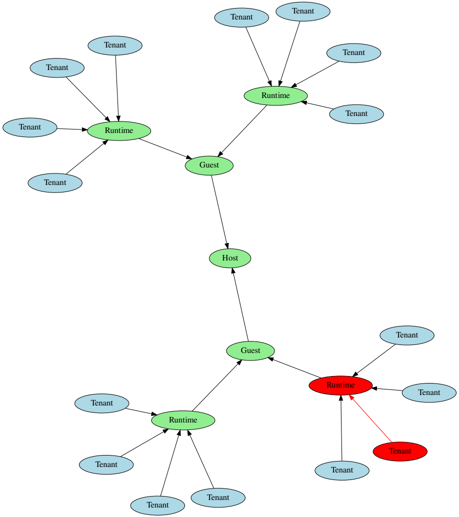


Figure 33: Compromise at depth=1

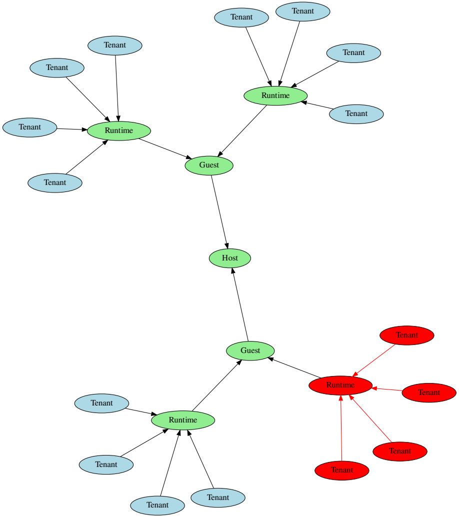


Figure 34: Blast radius, compromise depth=1

# Co-instantiation

- Density
  - Tenants / host
- Utilisation
  - Total Capacity
  - Tenant Occupancy
- Distribution
  - Allocation algorithm
  - Tenant Proximity
- Emphemerality
  - Maximum lifespan
  - Average lifespan

# Thanks everyone!

## Contact - Michiel Kalkman

- https://michielkalkman.com/
- https://au.linkedin.com/in/kalkmanmichiel

# Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)