Subverting Operating System Properties through Evolutionary DKOM Attacks

Mariano Graziano, Lorenzo Flore, Andrea Lanzi, Davide Balzarotti

Cisco Systems, Inc.
Universita' degli Studi di Milano
Eurecom

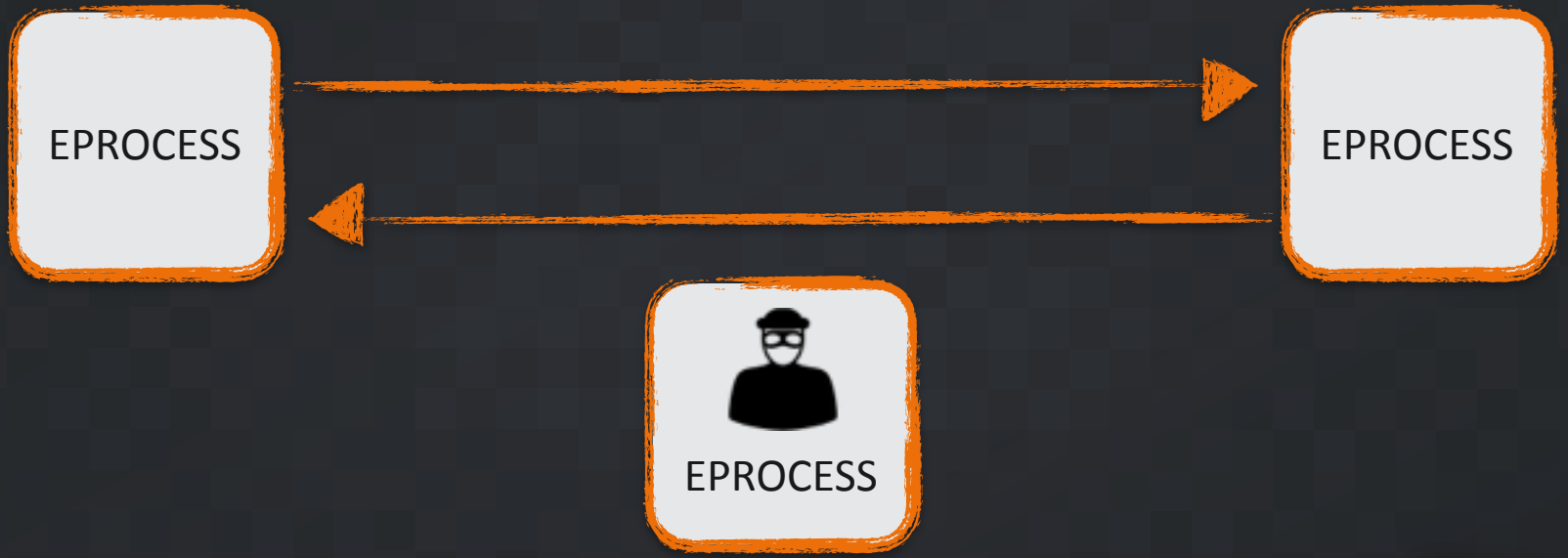
DIMVA 2016 - San Sebastian, Spain

TALOS

TRADITIONAL DKOM ATTACKS

TALOS

TRADITIONAL DKOM ATTACKS

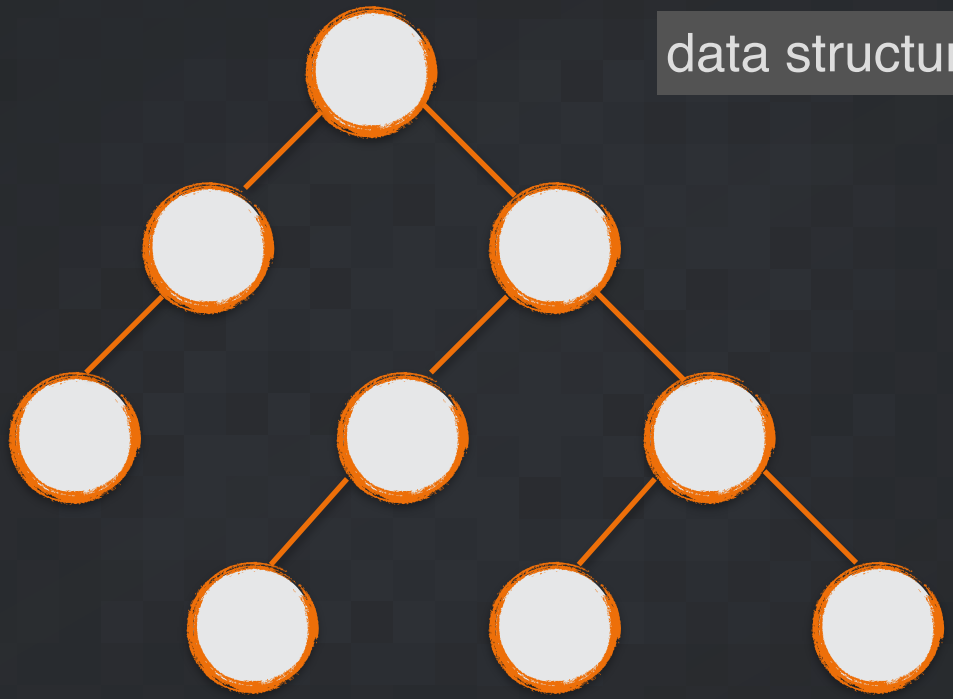


TALOS

TRADITIONAL DKOM DEFENSES

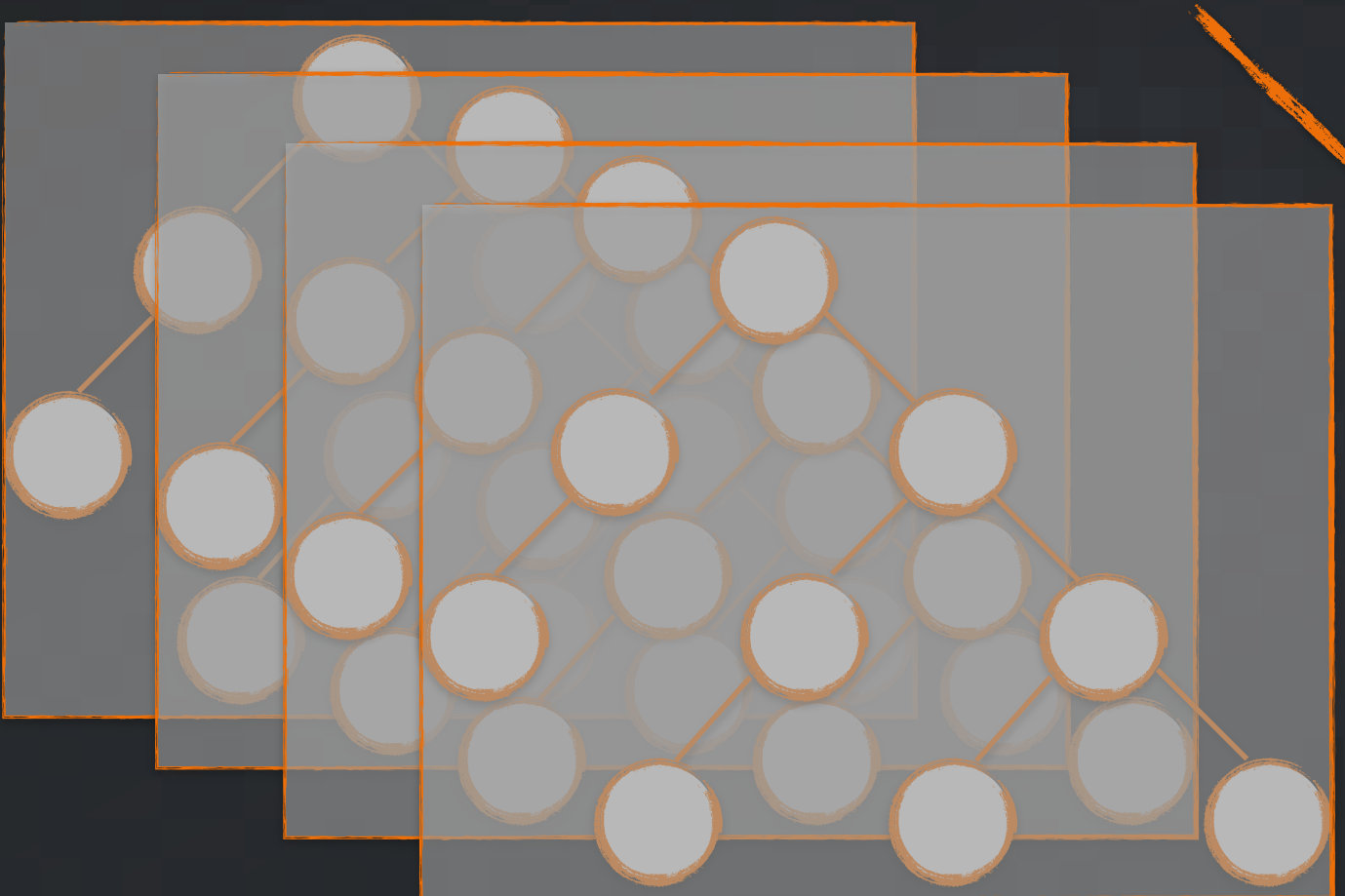
- Kernel data integrity solutions:
  - invariants
    - external systems
    - memory analysis
  - data partitioning

EVOLUTIONARY DKOM ATTACKS



data structure of interest

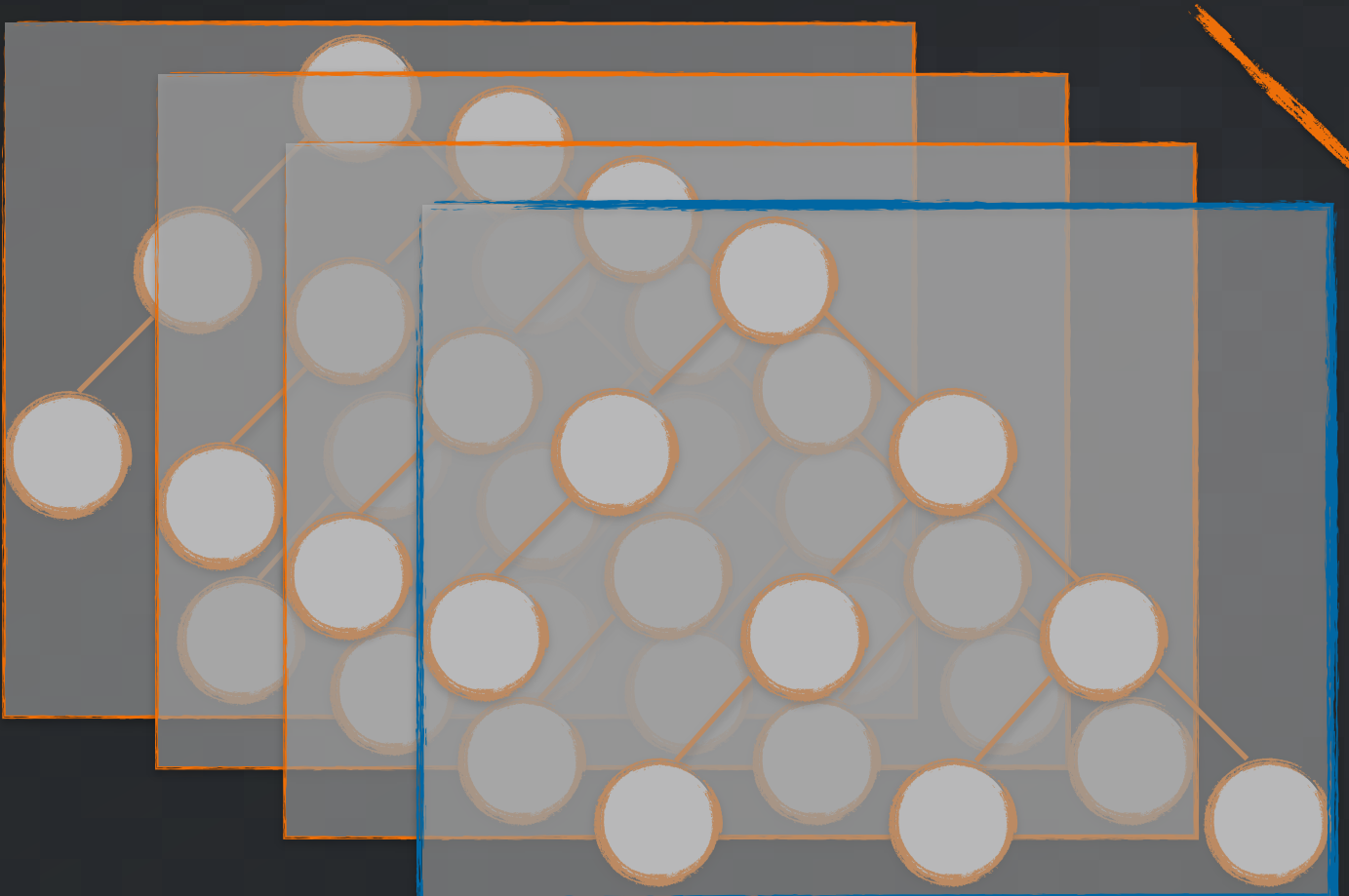
EVOLUTIONARY DKOM ATTACKS



Violation of a temporal property

TALOS

EVOLUTIONARY DKOM ATTACKS



Violation of a temporal property

the attack cannot be detected looking at a single snapshot

TALOS

STATE VS PROPERTY

- Traditional DKOM affects the state and are discrete
- Evolutionary DKOM (E-DKOM) affects the evolution in time of a given property and are continuous

TALOS

THREAT MODEL

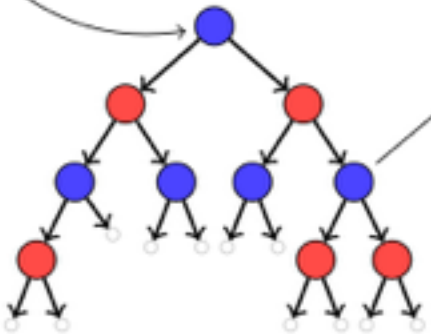
- Attacker has access to ring0
- Malicious code not detectable by current solutions
- Attacker cannot modify kernel code and attack the VMM

EXAMPLE: LINUX CFS SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```



```
struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```

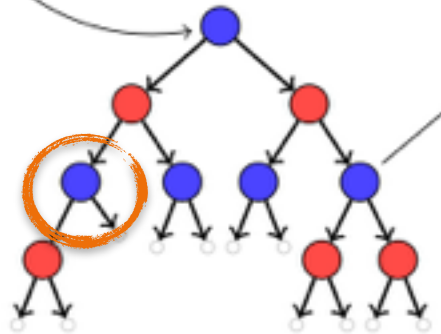
SUBVERTING THE SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```

```
struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```



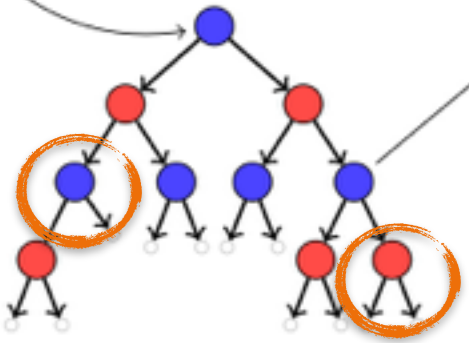
SUBVERTING THE SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```

```
struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
  struct rb_node *rb_left;
};
```



target

right most

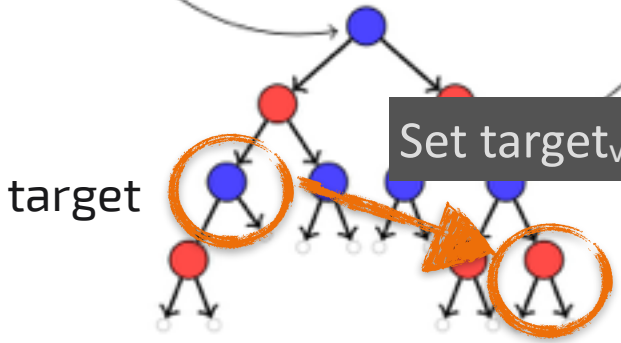
SUBVERTING THE SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```

```
struct rb_node{
  unsigned long rb_parent_color;
  struct rb_node *rb_right;
```



Set $target_{vruntime} > rightmost_{vruntime}$

target

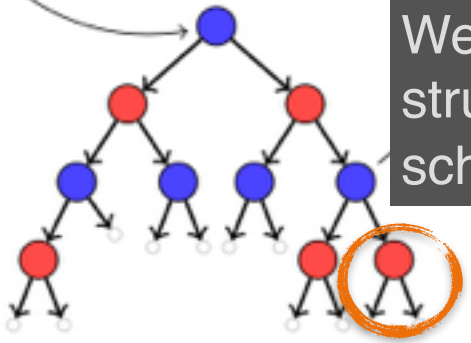
right most

SUBVERTING THE SCHEDULER

```
struct task_struct {
  volatile long state;
  void *stack;
  unsigned int flags;
  int prio, static_prio, normal_prio;
  const struct sched_class *sched_class;
  struct sched_entity se;
  ...
}
```

```
struct sched_entity {
  struct load_weight load;
  struct rb_node run_node;
  struct list_head group_node;
  ...
}
```

```
struct cfs_rq {
  ...
  struct rb_root tasks_timeline;
  ...
};
```



We affected the evolution of the data structure over time. We altered the scheduler property (fair execution).

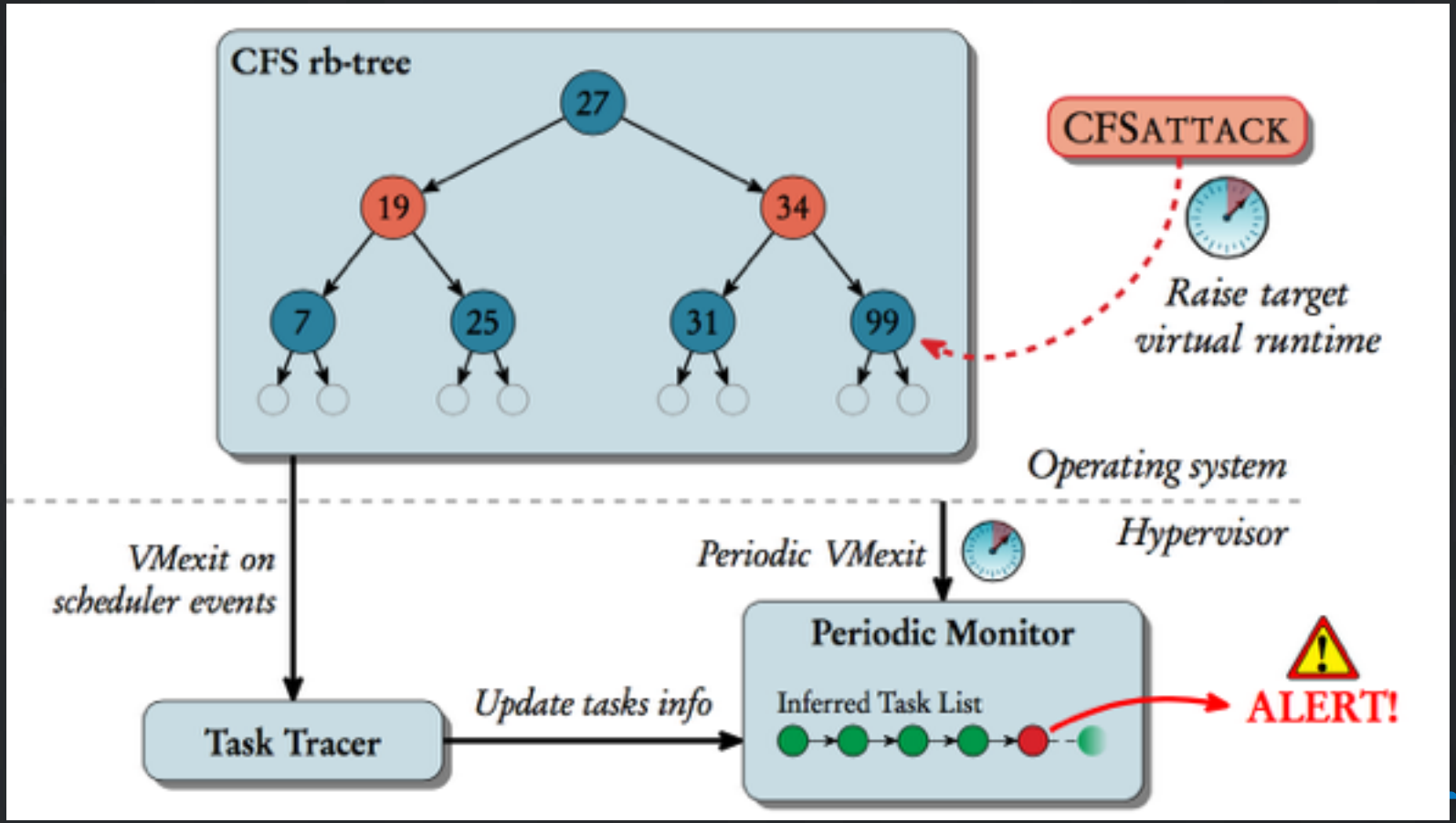
ATTACK EVALUATION

- Temporarily block an IDS or Antivirus
- Temporarily block Inotify

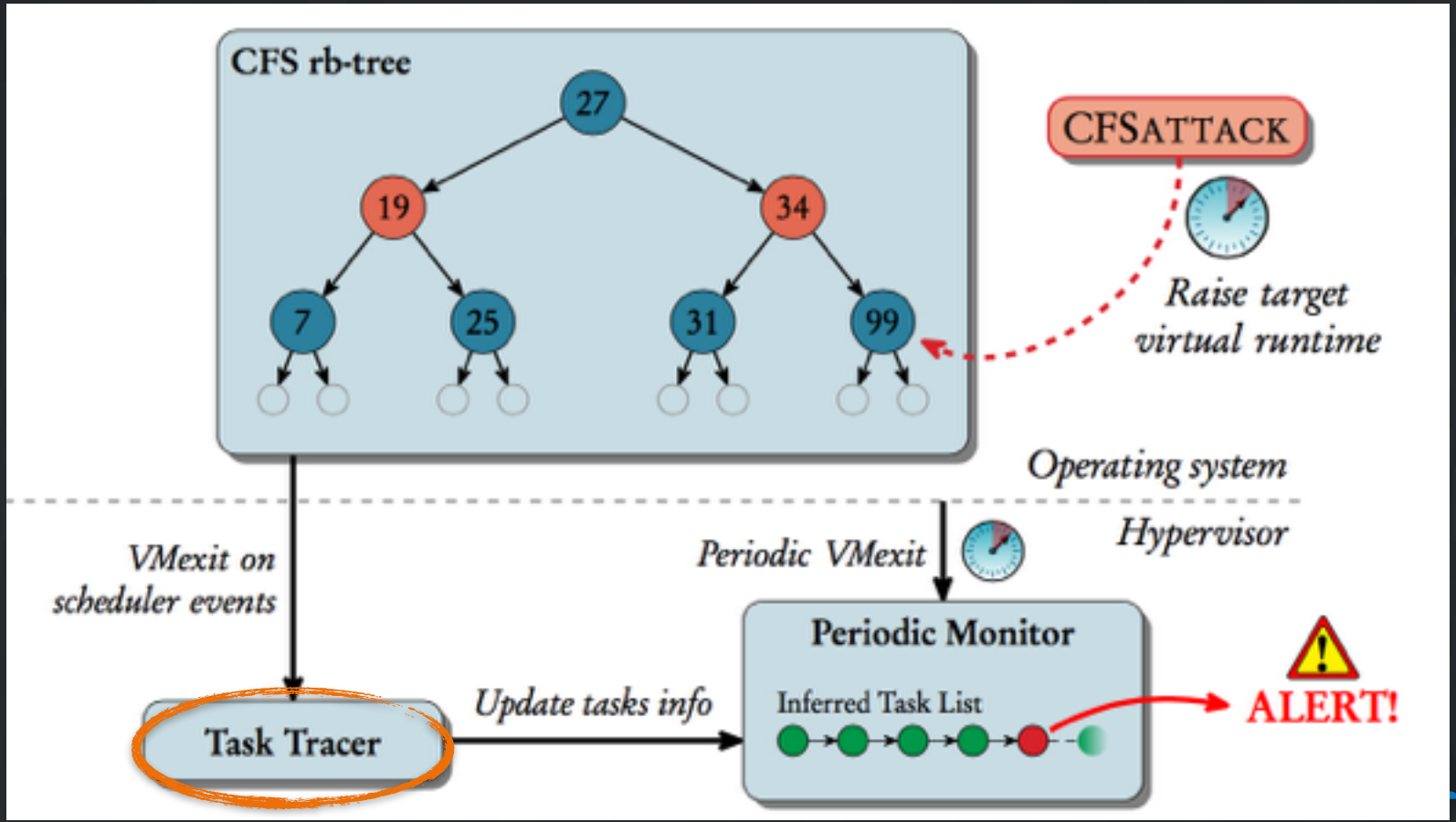
DEFENSES?

- Reference monitor that mimics the OS property:
  - OS specific
  - Difficult to generalize

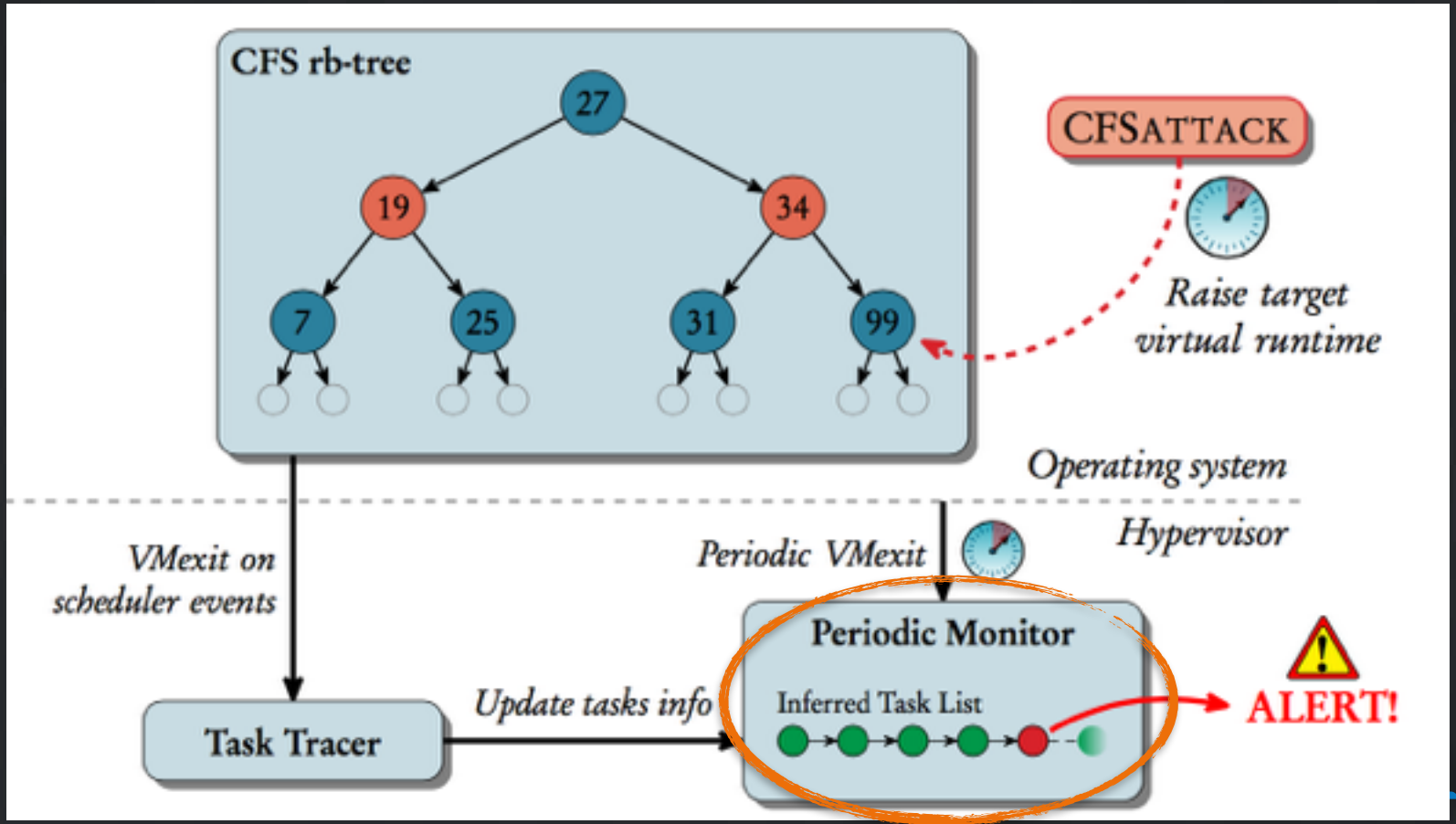
DEFENSE FRAMEWORK



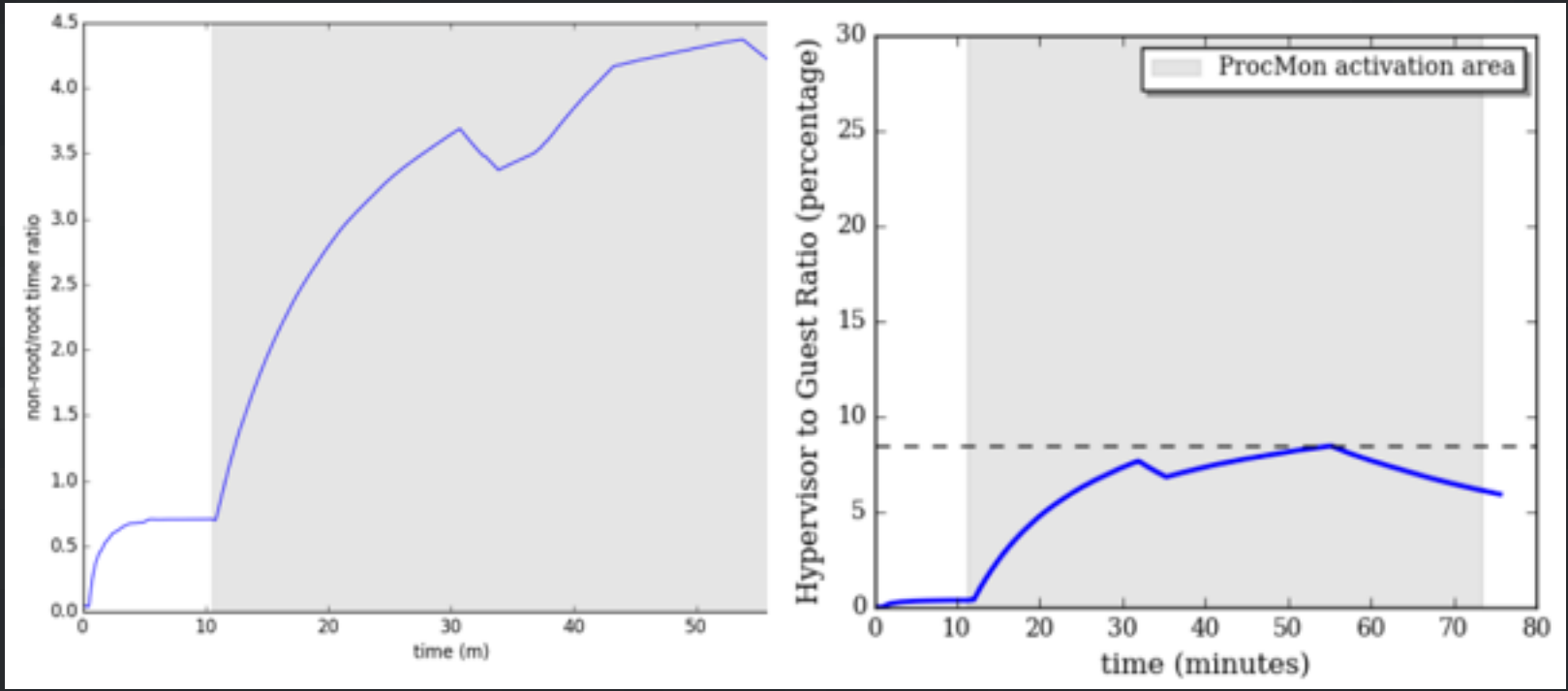
DEFENSE FRAMEWORK



DEFENSE FRAMEWORK



OVERHEAD



Normal operations

Stress test

TALOS

CONCLUSIONS

- New DKOM attack based on data structures evolution
- Experiment on the Linux CFS scheduler
- Defense solution based on hypervisor
- General mitigation/solution very hard

TALOS

QUESTIONS?

Mariano Graziano
magrazia@cisco.com
@emd3l