Toward the design and implementation of a language dedicated to Virtual Machine Introspection

Lionel HEMMERLE, Guillaume HIET, Frédéric TRONEL, Pierre WILKE, Jean-Christophe PREVOTET

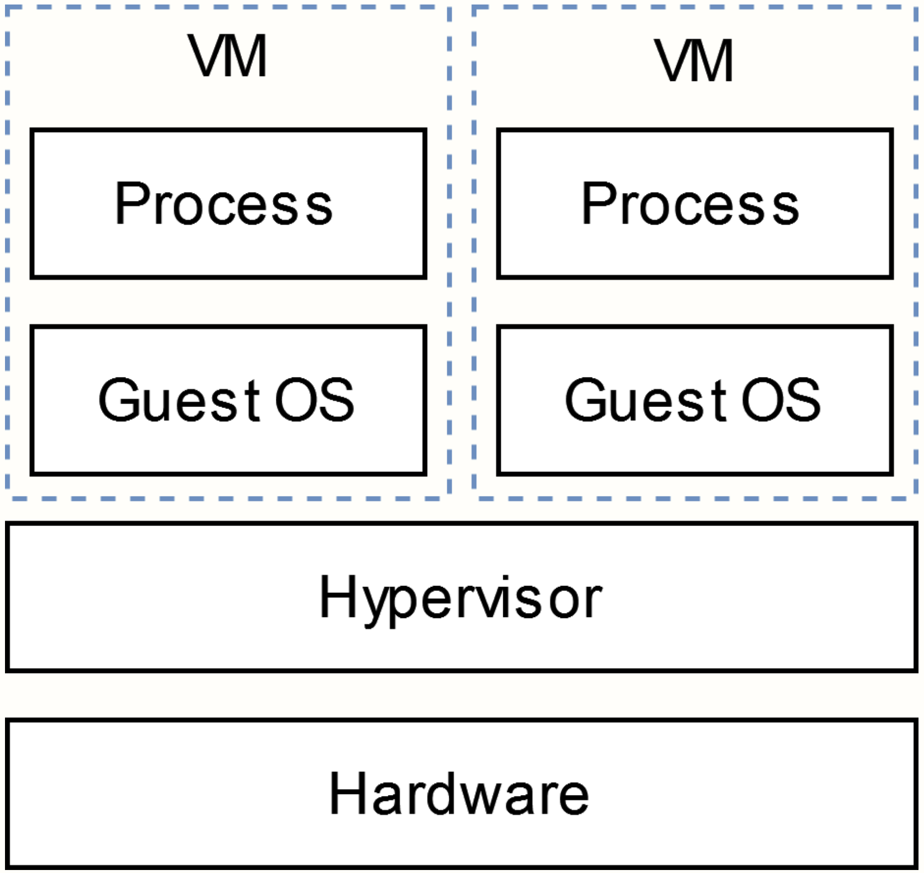
CentraleSupélec · Inria · Université de Rennes · CNRS · UMR IRISA

Context: Intrusion Detection

- **Two approaches:**
  - Network based Intrusion Detection System
    - Unable to detect some attacks (e.g., a local privilege escalation)
  - Host-based Intrusion Detection System
    - Rely on data sent by the OS to detect intrusions
- **How can we detect intrusions when the OS is compromised?**
  - Intruders can send false data to the IDS
- **We propose to use virtualization extensions**

Context: Virtualization

- **Virtualization extension:**
  - Allow a processor to run multiple Virtual Machines
  - A software component named hypervisor manages those VMs
- **Advantages:**
  - Each VM is isolated from others
  - Even if an attacker can control one VM, the hypervisor is still safe
  - The IDS can be placed in the hypervisor



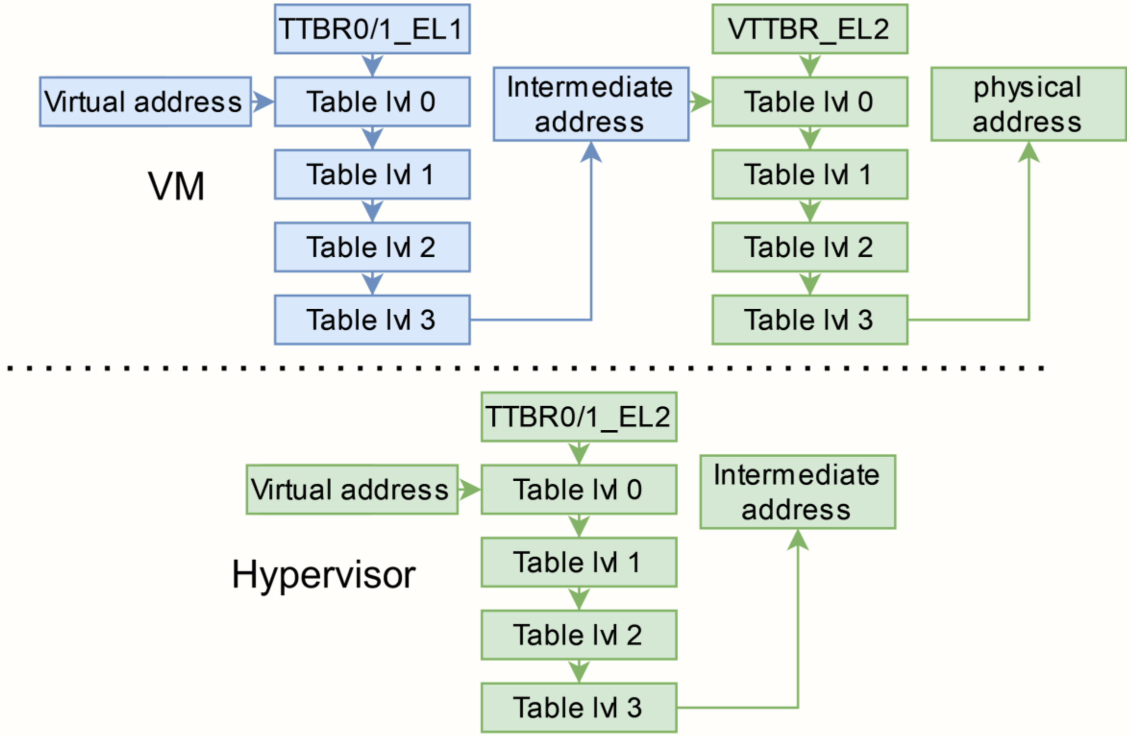
Context: Semantic gap

- **Semantic gap: how can we retrieve data from the VM memory**
  - How are they stored?
    - E.g., The format of the structures used by the kernel
  - Where are they stored?
    - Virtual and physical addresses
  - When are they updated?
    - Some changes are legitimate

- **Example: processes list**

Example: Where are data stored?

- **Address Translation mechanism in two steps**
  - One controlled by the VM
  - One controlled by the hypervisor
- **The IDS needs to do the whole process to retrieve data**



How to obtain data ?

- **Development of specific tool for each OS**
  - Fastidious to develop
- **Learning based approaches**
  - May fail to detect new behaviors
- **Cloning of the VM**
  - Time consuming process
- **Communication with the VM**
  - An intruder can send incorrect data to the hypervisor

Interesting approach: Hyperupcalls

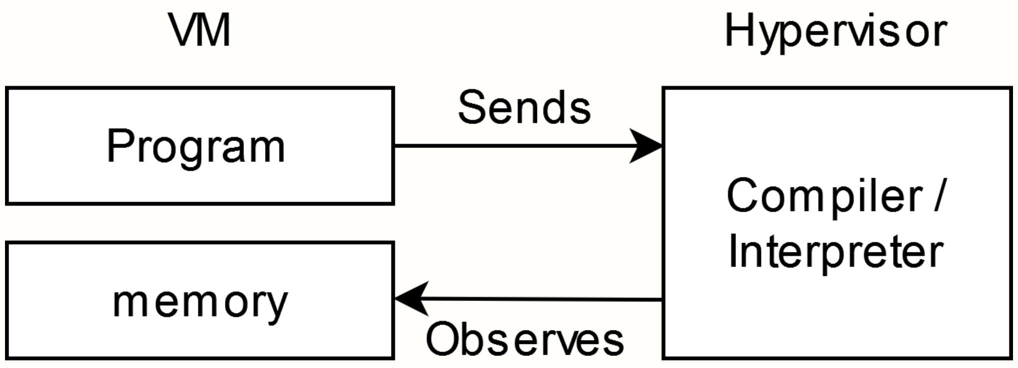
- **VM can send eBPF programs to the hypervisor**
  - The hypervisor can execute those programs
  - They allow the hypervisor to get information about resources used by the VM
  - Allow a better use of hardware resources by the hypervisor

- **Drawbacks**
  - Not designed for security purposes
  - Programs can be sent at any time by the VM
  - An attacker can change the memory structure layout to disable those programs

Michael Wei et Nadav Amit. « Leveraging Hyperupcalls To Bridge The Semantic Gap: An Application Perspective ». IEEE Data Eng. Bull. (2019).

Our objective

- **Create a language dedicated to VM Introspection**
  - VMs send some programs
  - Hypervisor compiles and execute those programs
  - They can raise an alert when an attack is detected



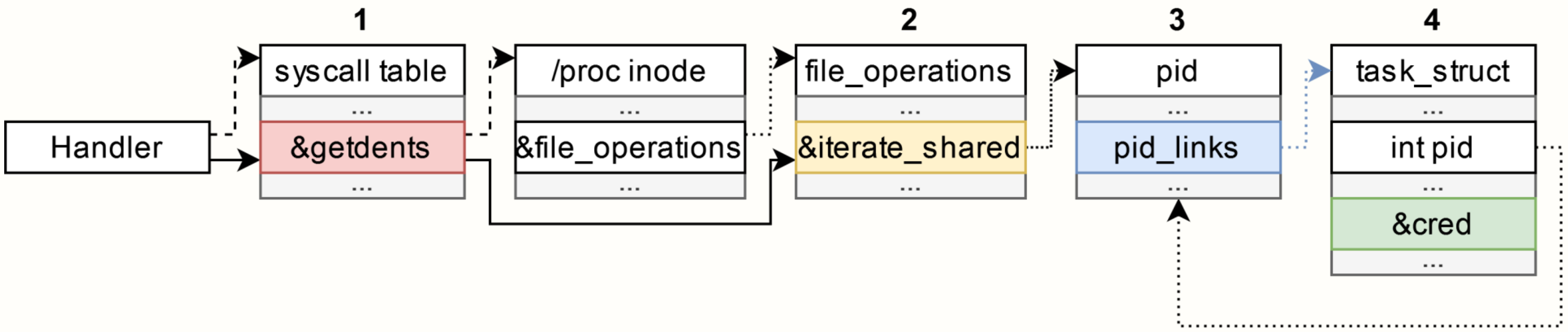
Implementation

- **Hypervisor modification**
  - Read/Write detection in VM memory
  - Detection of the execution of an instruction at a given address
  - Hypercall allowing a VM to send programs written in our language
  - Small interpreter running the programs sent by the VM (still WIP)
    - Can be used to detect two rootkits we have implemented

- **Creation of different rootkits**

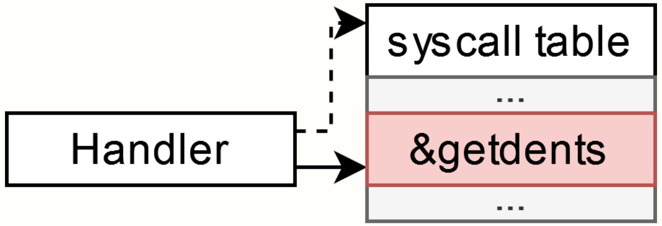
Rootkits

1. Syscall table modification
2. Virtual file system modification
3. Modification of structures `pid` and `task_struct`
4. Modification of structures `cred`



Rootkits

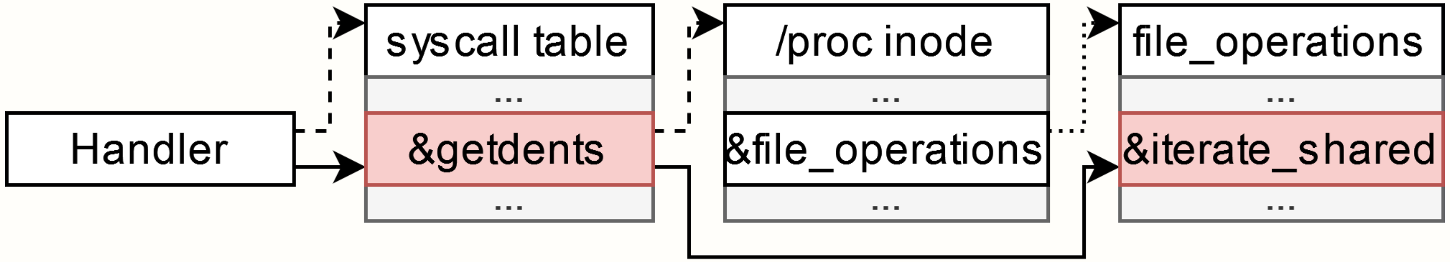
- **Syscall table modification**
  - A list of functions corresponding to every system call is stored in the `syscall table`
  - `getdents` is called to list the existing processes
- **Attack: change of a function pointer in the syscall table**
- **Detection: the syscall table should not be changed after boot**



Rootkits

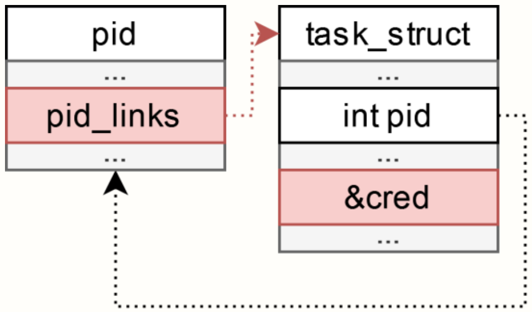
- **Syscall table modification**
  - A list of functions corresponding to every system call is stored in the `syscall table`
  - `getdents` is called to list the existing processes
- **Attack: change of a function pointer in the syscall table**
- **Detection: the syscall table should not be changed after boot**

Some other structures may also be changed by the attacker



Rootkits

- **Modification of dynamic structure**
  - `pid`: structure used by iterate_shared to iterate on processes
    - May be created or destroyed dynamically (e.g., when a new process is created)
  - `cred`: store the credentials and permissions of a given process
    - May be changed dynamically (e.g., setreuid syscall)
- **Attack: illegal changes of those structures**
- **Detection: needs to differentiate legitimate and illegitimate changes**



Language

- **Currently WIP**

- **Programs are sent by the VM and interpreted by the hypervisor**

- **List of useable events:**
  - Read/write in memory
  - Write in a system register
  - Call of a given function in the VM kernel

Language

x ∈ Var

m ∈ MapId

k ∈ Key

u ∈ Uint64

f ∈ FunName ::= `string`

r ∈ Reg ::= `pc` | `reg(i)`

e ∈ Expr ::= `cst` u | `var` x
| `binop` bop e1 e2 | `unop` uop e
| `lookup` m k
| `VMreg` r | `VMmem` α
| `event`

k ∈ Kind ::= `Intermediate` | `Virtual`

α ∈ Addr ::= Kind × Expr

ı ∈ Range ::= Kind × Expr × Expr

a ∈ Access ::= R | W

ev ∈ Event ::= `mem access` a ı
| `reg access` r
| `break` α

i ∈ Instr ::= `skip` | i1; i2 | x := e | `loop` n i
| `if` e `then` i1 `else` i2
| `delete` m k | `update` m k e
| `alert`
| x ← `add listener` ev f
| `remove listener` x

c ∈ Cmd ::= `kernel pagetable` u
| m := `create map`
| `fundef` f i
| `dofun` f
| `stop trusting me`

Security constraints

- **Sending programs**
  - An attacker could try to send its programs
- **Countermeasure**
  - The VM is initially safe
  - The VM sends a specific signal after the send of its last program
  - Any program send after that signal will be ignored by the hypervisor

Security constraints

- **Input manipulation**
  - An attacker can change programs inputs (VM Memory)
    - Ex: they can create new processes, …
  - It can try to use bugs in existing programs to compromise the whole hypervisor

- **Countermeasure**
  - Memory accesses are limited to maps (R/W) and VM memory (R)
  - No infinite loops
  - Raise an alert if the number of registered events for a VM reaches a threshold
  - Protect the structures and registers used by the address translation mechanism

Program example

```
main() {
    on_call(sys_fork, protect_task_struct)
    on_call(sys_exit, remove_task_protections)
}
protect_task_struct() {
    task = find_task_struct()
    handler = on_write(task.cred, sizeof(struct cred), protect_cred)
    store(process_map, task.pid, handler)
}
remove_task_protections() {
    task = find_task_struct()
    disable(get(process_map, task.pid))
}
```

Conclusion & future work

- **Work done**
  - Modification of a hypervisor to detect events in a VM
  - Hypercall allowing a VM to send programs to the hypervisor
  - Small interpreter int the hypervisor allowing to detect some rootkits in a VM
- **Future work**
  - Continue the development of a simple interpreter (then a compiler)
    - They should ensure than security constrains are respected
  - Create programs in our language to detect the four implemented rootkits
  - Benchmark the performances of an introspected VM