Semantics-Driven Introspection in a Virtual Environment

F. Baiardi[1] D. Maggiari[1] D. Sgandurra[2] F. Tamberi[2]

[1]Polo G. Marconi - La Spezia, University of Pisa

[2]Department of Computer Science, University of Pisa

IAS Conference, 2008

Outline

1. Problem
   - Sense of Self
2. Overall Architecture
   - Assertions and System Calls
   - Virtualization
   - Overall Architecture
3. Evaluation
   - Performance
4. Conclusion
   - Results and Future Works

Attacks Against the Self

- Protecting a process from attacks that alter the intended behavior of the executed program.
- We want to preserve the original semantics of the program.
- We are not interested in logic errors, such as:
  - authentication errors (weak passwords);
  - malicious behavior.

Buffer Overflow

- The program stores more data into a buffer than the memory space reserved for it.
- The attacker may overwrite data that controls the program's flow:
  - control-hijacking attack: the attacker can diverge the control flow;
  - malicious code is executed.
- If the program has the rights of invoking any system call, the attacker gains control of the system.

A Sense of Self for Processes

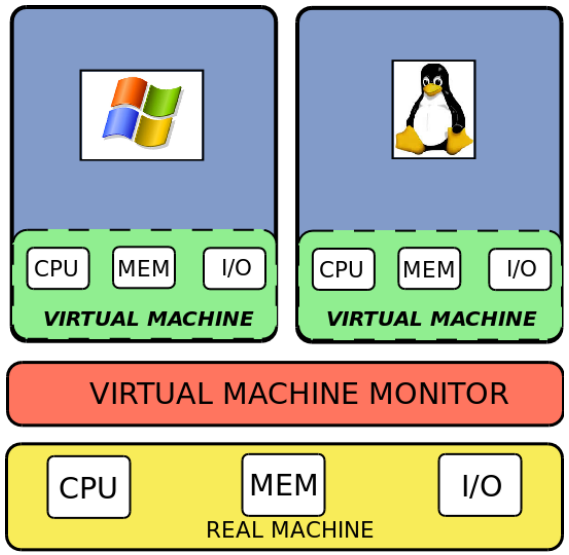
- Notion of process self: the program that the process executes.
- Based on traces of system calls:
  - dynamic analysis: Forrest et al;
  - static analysis: Wagner and Dean.
- Assumption: a process can execute security critical operations only through system calls.
- Denial of service attacks are still possible!

General Approach

- Find system call sites in the program's source code:
  - their return address.
- Generate an invariant for each system call:
  - relate values of programs variables and of system call parameters.
- At run-time, access the memory of the monitored process to evaluate an invariant each time the process issues a system call.
- Exploit virtualization technology.

Virtualization

- Virtual Machines (VMs): execution environments that emulate, at software, the behavior of the underlying physical machine.
- A standard machine can support several VMs.



Architecture

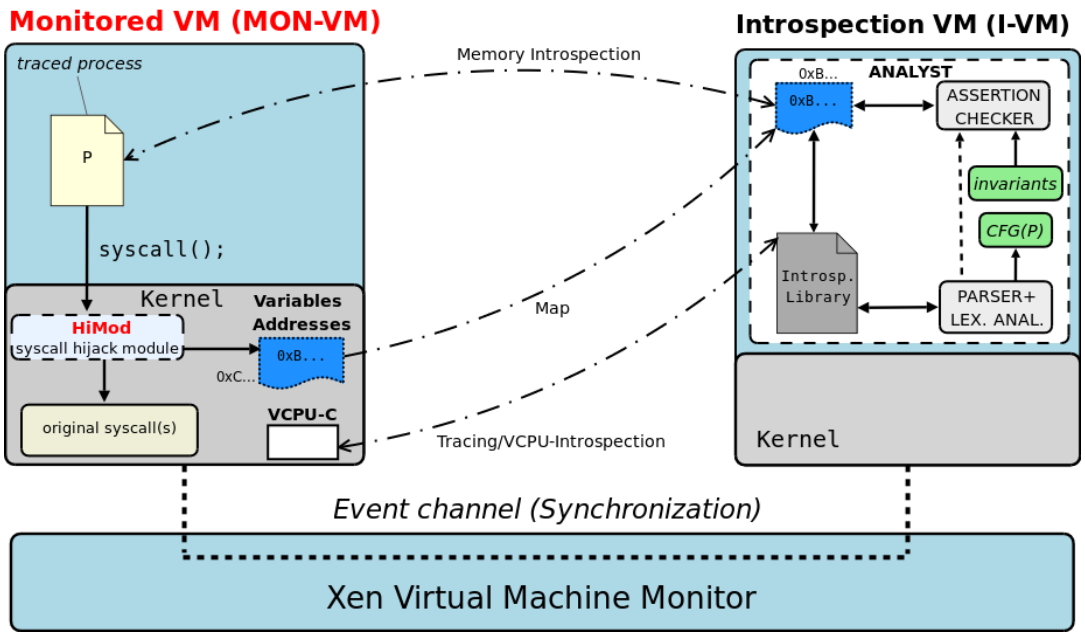
Based on two virtual machines:

- the Monitored VM (Mon-VM), i.e. the VM executing the process to be monitored;
- the Introspection VM (I-VM), i.e. the VM monitoring the process through virtual machine introspection:
  - Assertion Checker: to evaluate invariants;
  - Introspection Library: to access the memory of the monitored VM.

Monitored VM

It runs the monitored process.

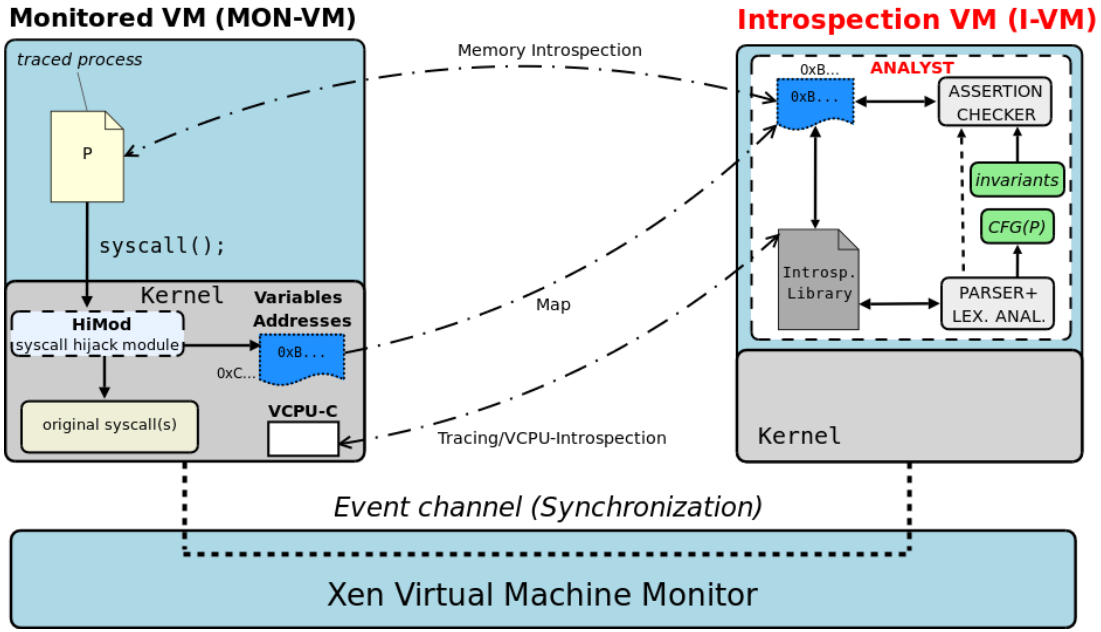
- HiMod: Linux Kernel Module to hijack system calls.
- Only a subset of system calls is traced: most critical ones.



Introspection VM

It runs the monitoring systems and applies the consistency checks.

- It exploits the Introspection Library to access the monitored VM.
- Assertion Checker evaluates invariants.



Introspection Library

- The Introspection Library is invoked by the Assertion Checker whenever the monitored process issues a system call.
- Memory Introspection, to access the memory of a monitored VM both at the user and at the kernel level.
- VCPU-Context Introspection, to retrieve the state of the monitored VM's virtual processor.

Evaluating Invariants

- To detect non-control-data attacks and mimicry attacks.
- Attacks based upon parameters of system calls.
- Assertions can be deduced by using dynamic tools (e.g. Daikon) or by a static analysis (e.g. CodeSurfer);
- Currently, we use a combination of Daikon, CodeSurfer and programmer-provided assertions.

Evaluating Invariants

- The kernel of the Monitored VM transfers control to the Introspection VM every time the process invokes a system call.
- The Introspection VM freezes the execution of the Monitored VM.
- The Assertion Checker exploits the Introspection Library to:
  - retrieve the current return address of the process;
  - retrieve the values of the some variables;
  - evaluate the invariant.

Evaluating Invariants

- The input is a set of invariants of the form:

```
[PC, {var name: addr: type}, {expr on vars}]
```

  - `PC` is the program counter (return addr) paired with a system call;
  - `{var name: addr: type}` is a set of variable names, their virtual address and their type;
  - `{expr on vars}` is a set of relations among variables.

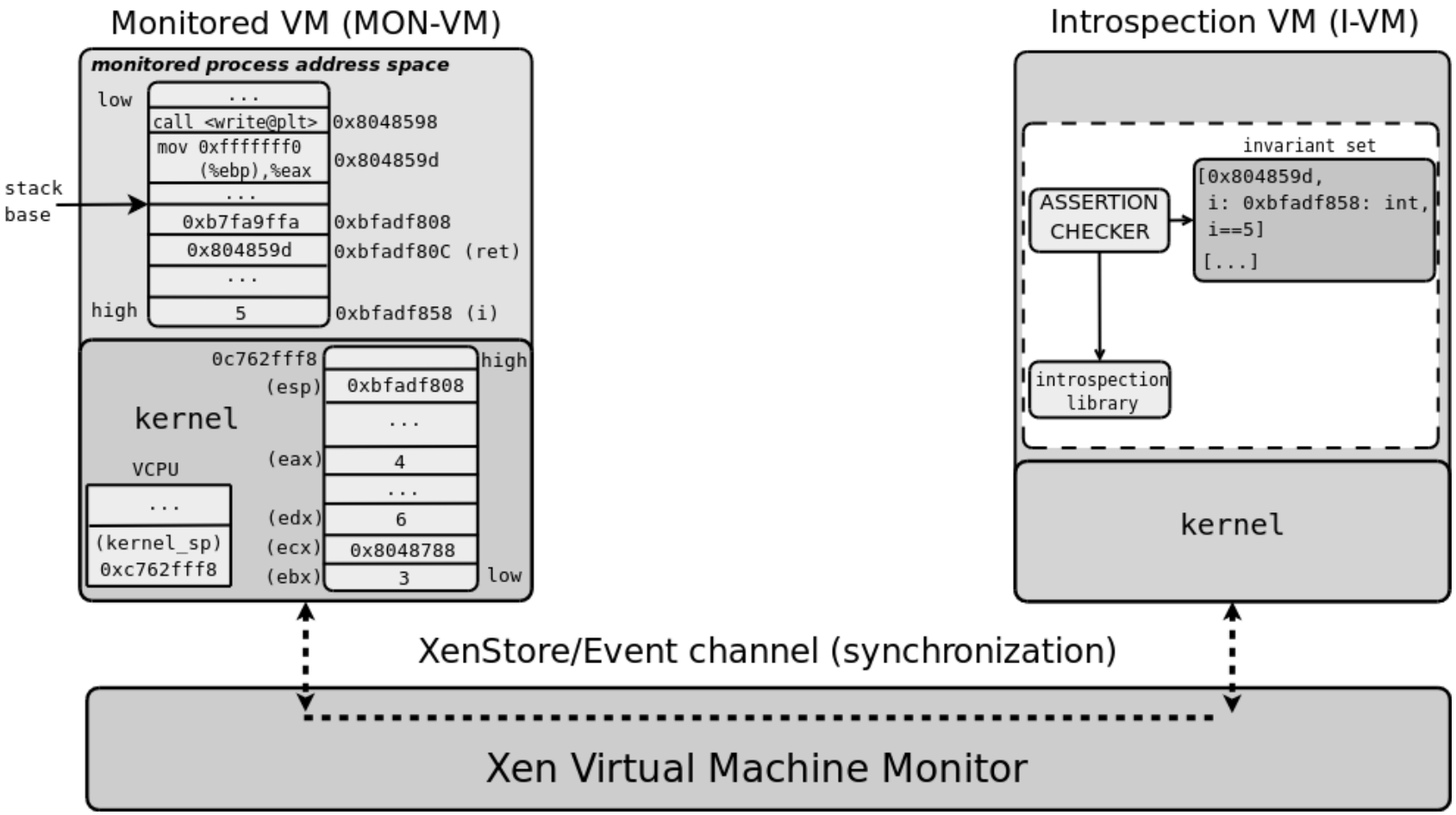
Examples of Invariants

- Parameters assertions:
  - data-flow relations among parameters of distinct calls;
  - e.g. the file descriptor in a `read` is the result of a previous `open`.
- File assertions:
  - prevent symlink and race condition attacks,;
  - e.g. real file-name of a file descriptor belongs to a known directory.

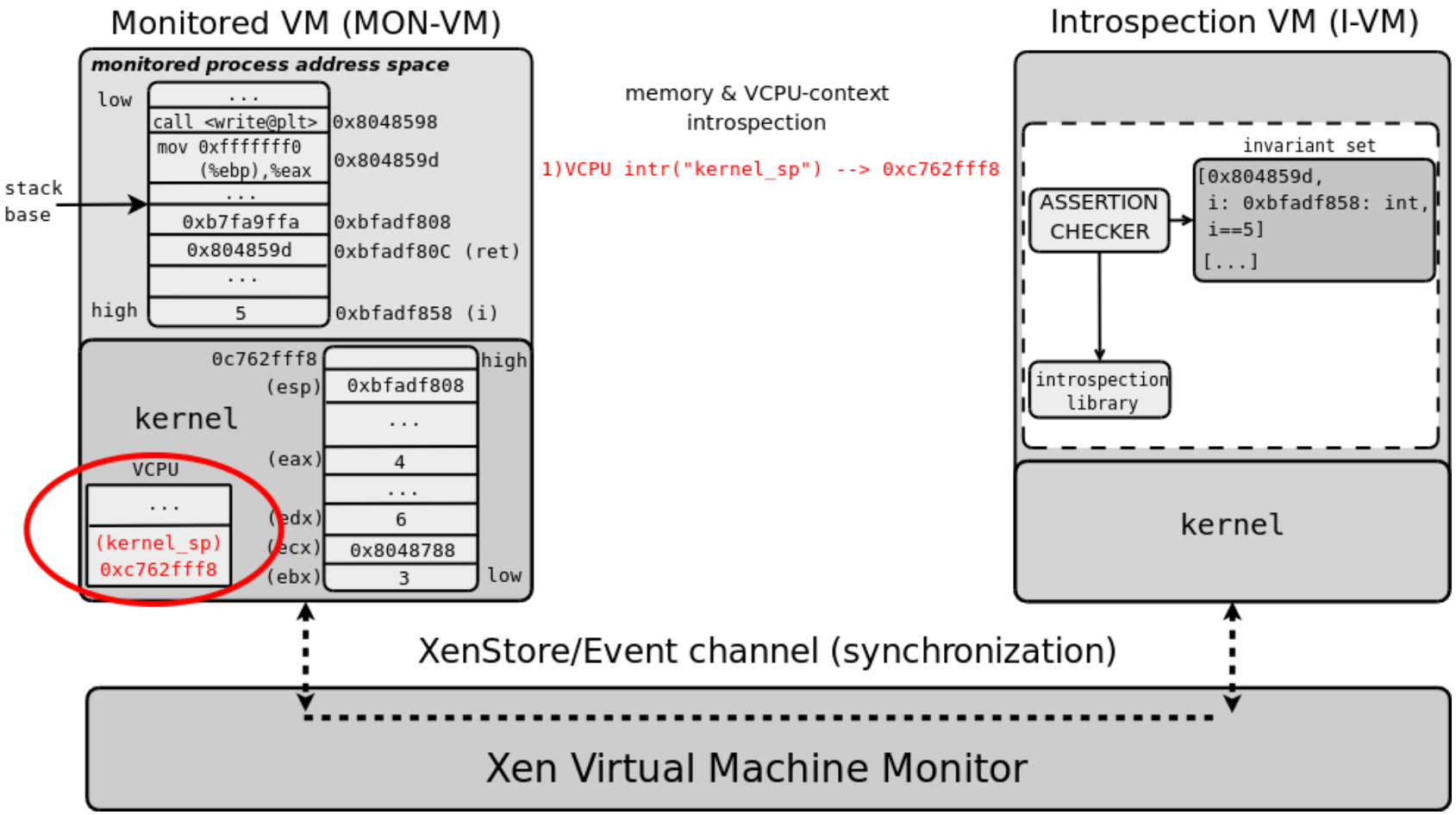
Examples of Invariants

- Buffer length assertions:
  - length of a string passed to a vulnerable function is not larger than the local buffer.
- Conditional statements assertions:
  - prevent *impossible paths* by relating a system call and the expression in the guard of a conditional statement:
    - e.g: `syscall`$_1$ `if(uid == 0) then syscall`$_2$ `else syscall`$_3$, pair the assertion `uid == 0` with `syscall`$_2$.

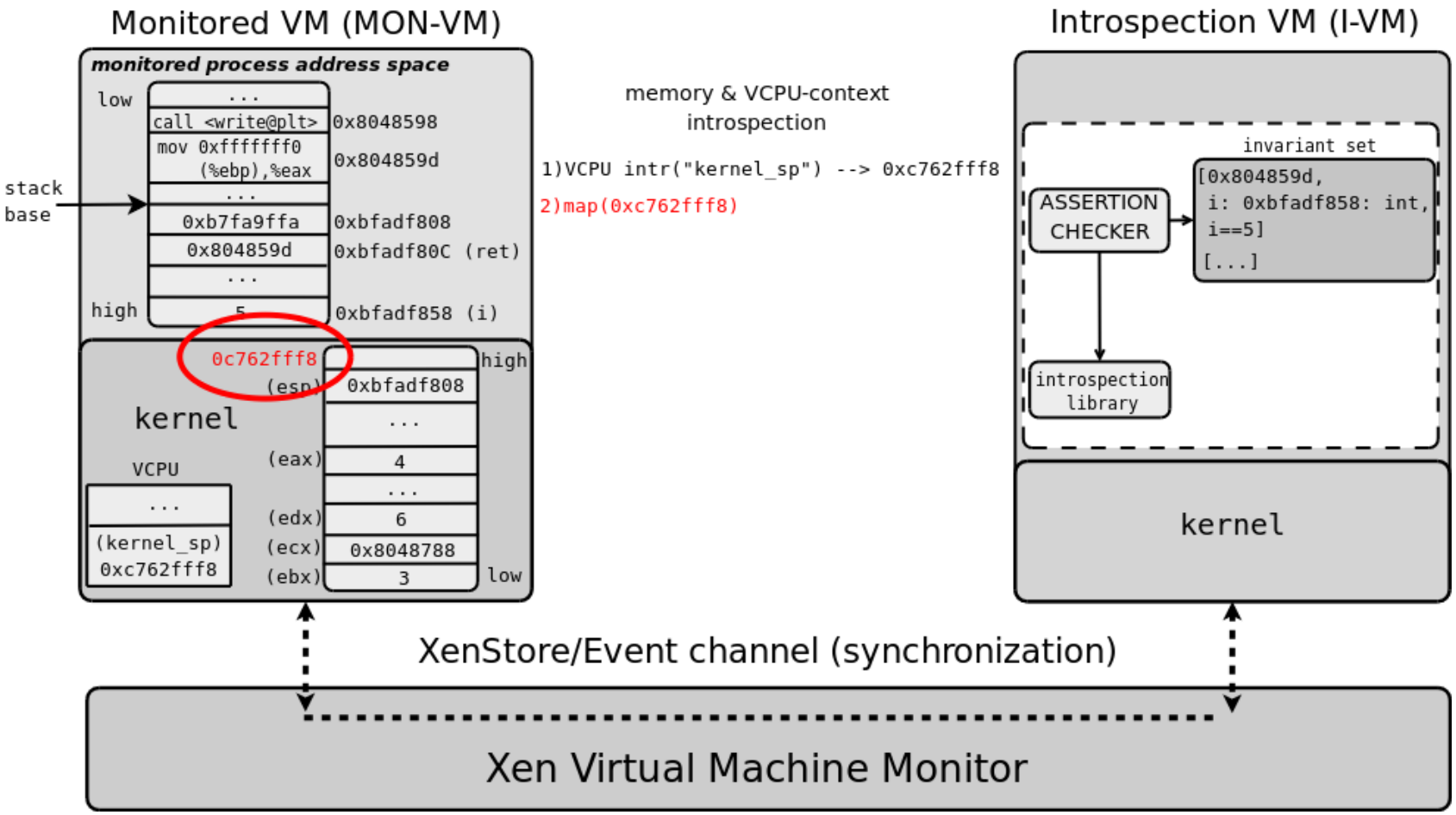
Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
stack base
...
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
...
high
5
0xbfadf858 (i)
0c762fff8
(esp)
high
0xbfadf808
kernel
...
(eax)
4
VCPU
...
(edx)
6
(kernel_sp)
0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
Introspection VM (I-VM)
invariant set
ASSERTION CHECKER
[0x804859d,
i: 0xbfadf858: int,
i==5]
[...]
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

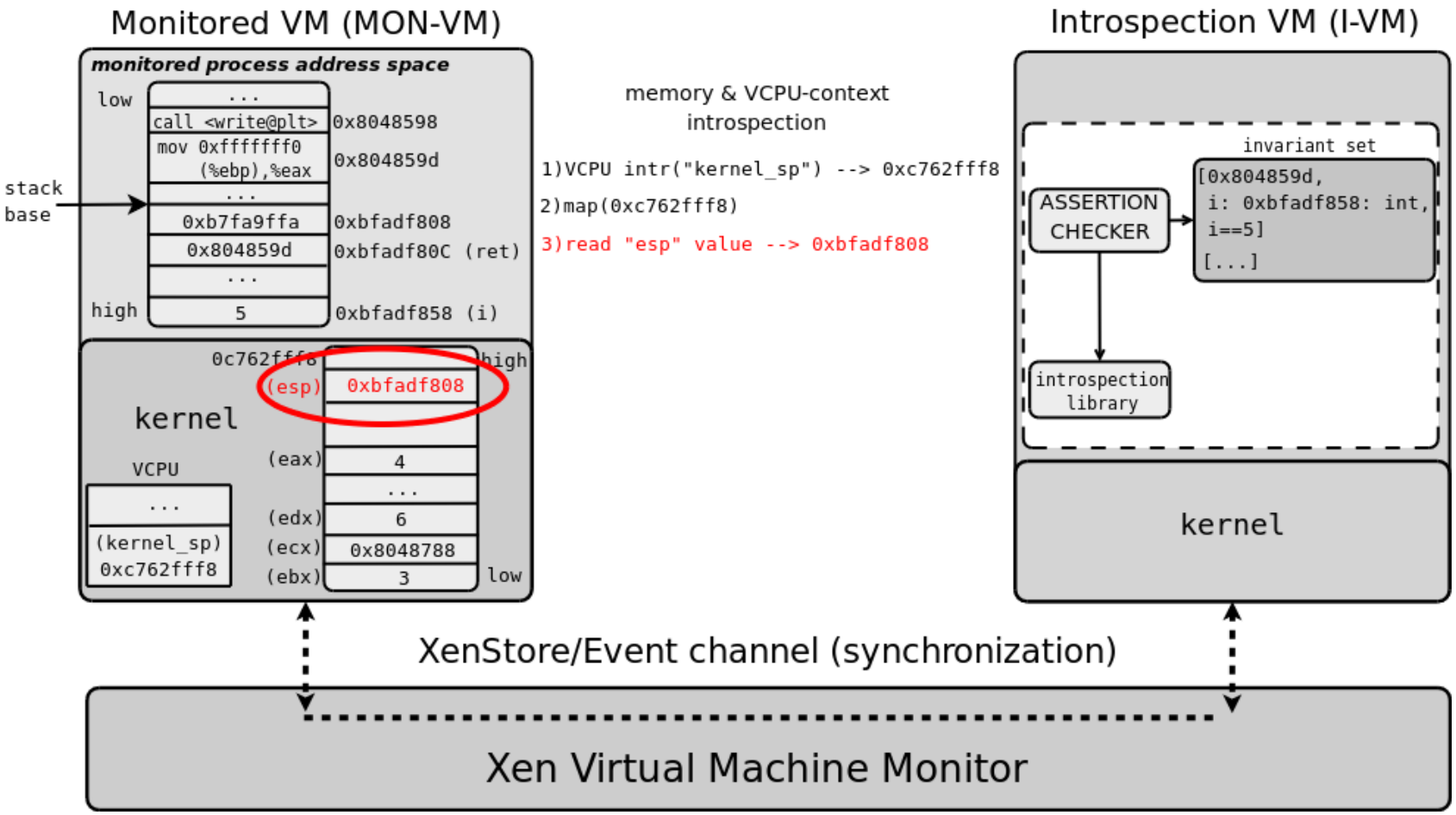
Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
...
stack base
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
...
high
5
0xbfadf858 (i)
0c762fff8 (esp)
high
0xbfadf808
kernel
...
(eax)
4
VCPU
...
(edx)
6
(kernel_sp) 0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
memory & VCPU-context introspection
1)VCPU intr("kernel_sp") --> 0xc762fff8
Introspection VM (I-VM)
invariant set
ASSERTION CHECKER
[0x804859d, i: 0xbfadf858: int, i==5]
[...]
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

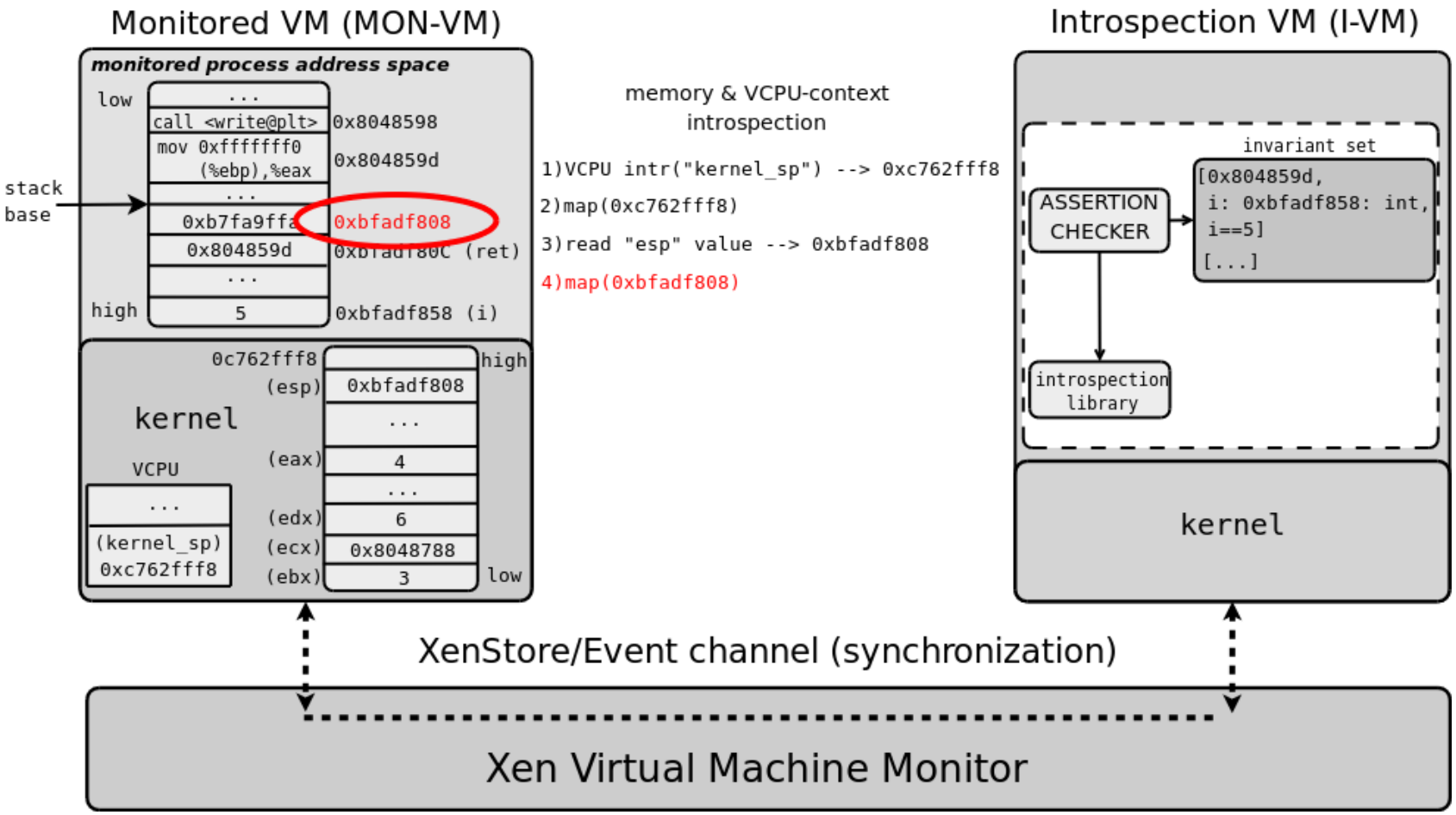
Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
...
stack base
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
...
high
5
0xbfadf858 (i)
0c762fff8
(esp)
high
0xbfadf808
kernel
...
(eax)
4
VCPU
...
...
(edx)
6
(kernel_sp)
0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
memory & VCPU-context introspection
1)VCPU intr("kernel_sp") --> 0xc762fff8
2)map(0xc762fff8)
Introspection VM (I-VM)
invariant set
ASSERTION CHECKER
[0x804859d, i: 0xbfadf858: int, i==5]
[...]
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

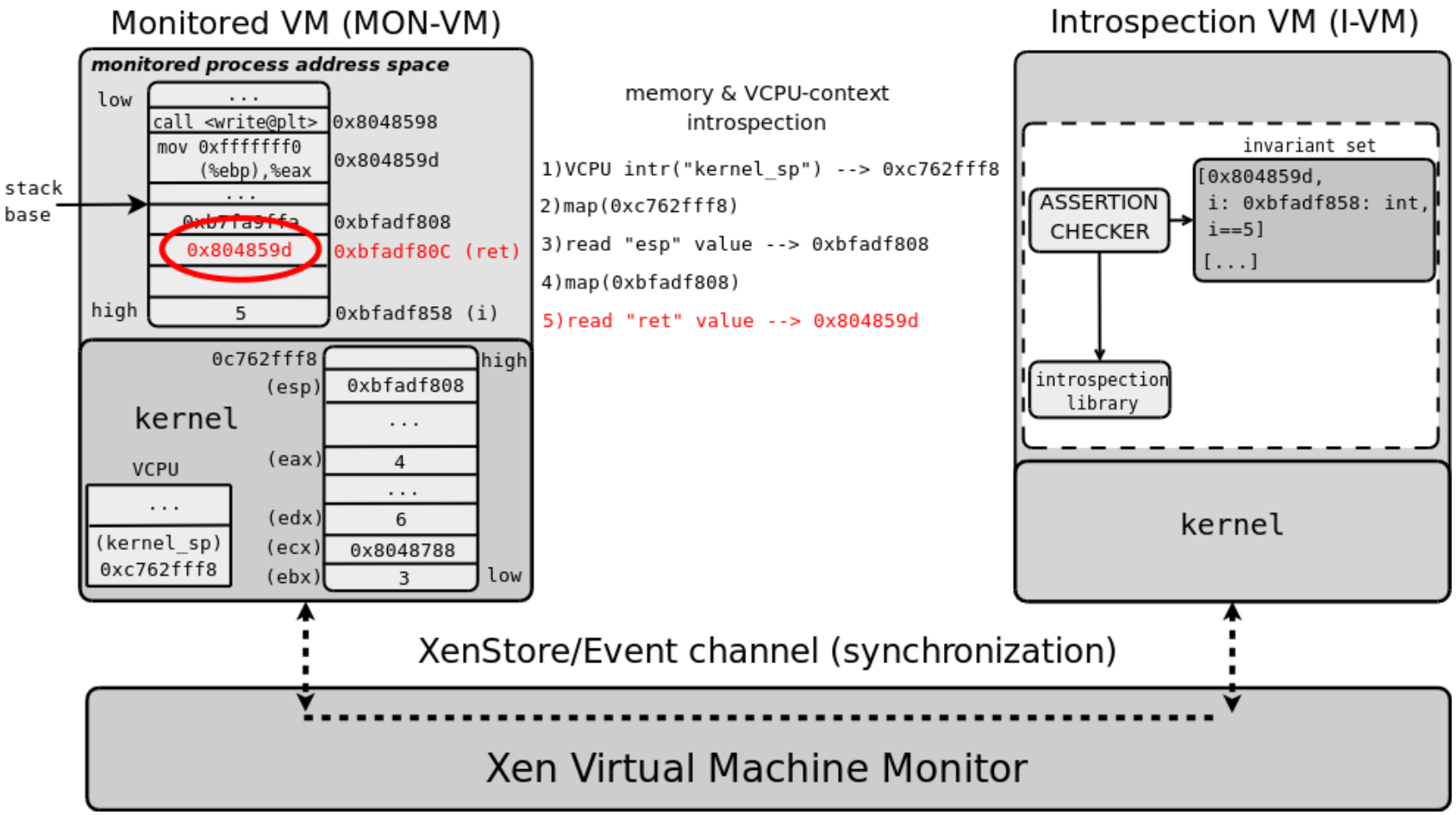
Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
...
stack base
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
...
high
5
0xbfadf858 (i)
0c762fff8
high
(esp)
0xbfadf808
kernel
(eax)
4
...
VCPU
...
(edx)
6
(kernel_sp)
0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
memory & VCPU-context introspection
1)VCPU intr("kernel_sp") --> 0xc762fff8
2)map(0xc762fff8)
3)read "esp" value --> 0xbfadf808
Introspection VM (I-VM)
invariant set
ASSERTION CHECKER
[0x804859d, i: 0xbfadf858: int, i==5]
[...]
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

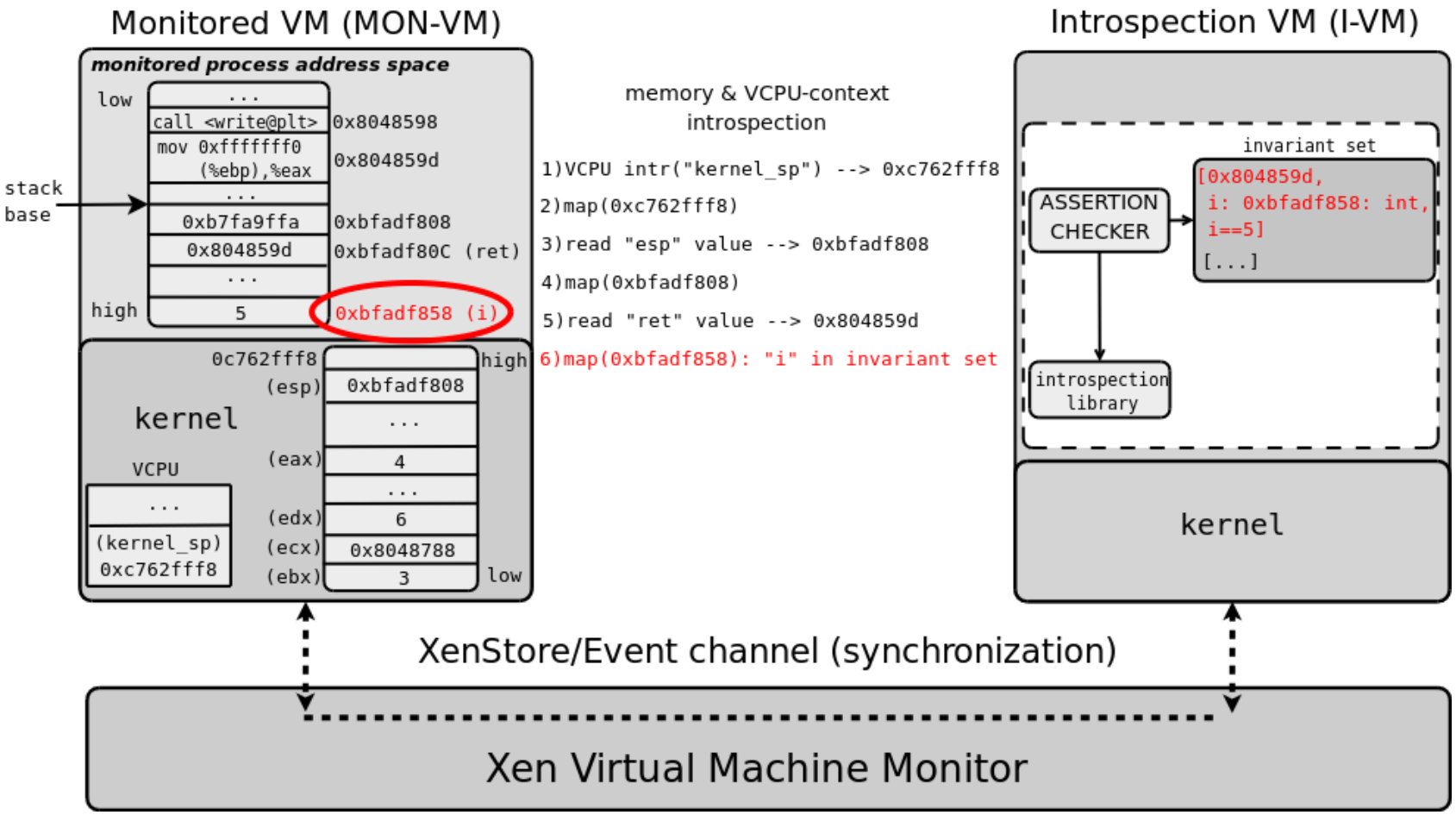
Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
stack base
...
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
...
high
5
0xbfadf858 (i)
0c762fff8
(esp)
high
0xbfadf808
kernel
...
(eax)
4
...
VCPU
...
(edx)
6
(kernel_sp)
0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
memory & VCPU-context introspection
1)VCPU intr("kernel_sp") --> 0xc762fff8
2)map(0xc762fff8)
3)read "esp" value --> 0xbfadf808
4)map(0xbfadf808)
Introspection VM (I-VM)
invariant set
ASSERTION CHECKER
[0x804859d,
i: 0xbfadf858: int,
i==5]
[...]
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

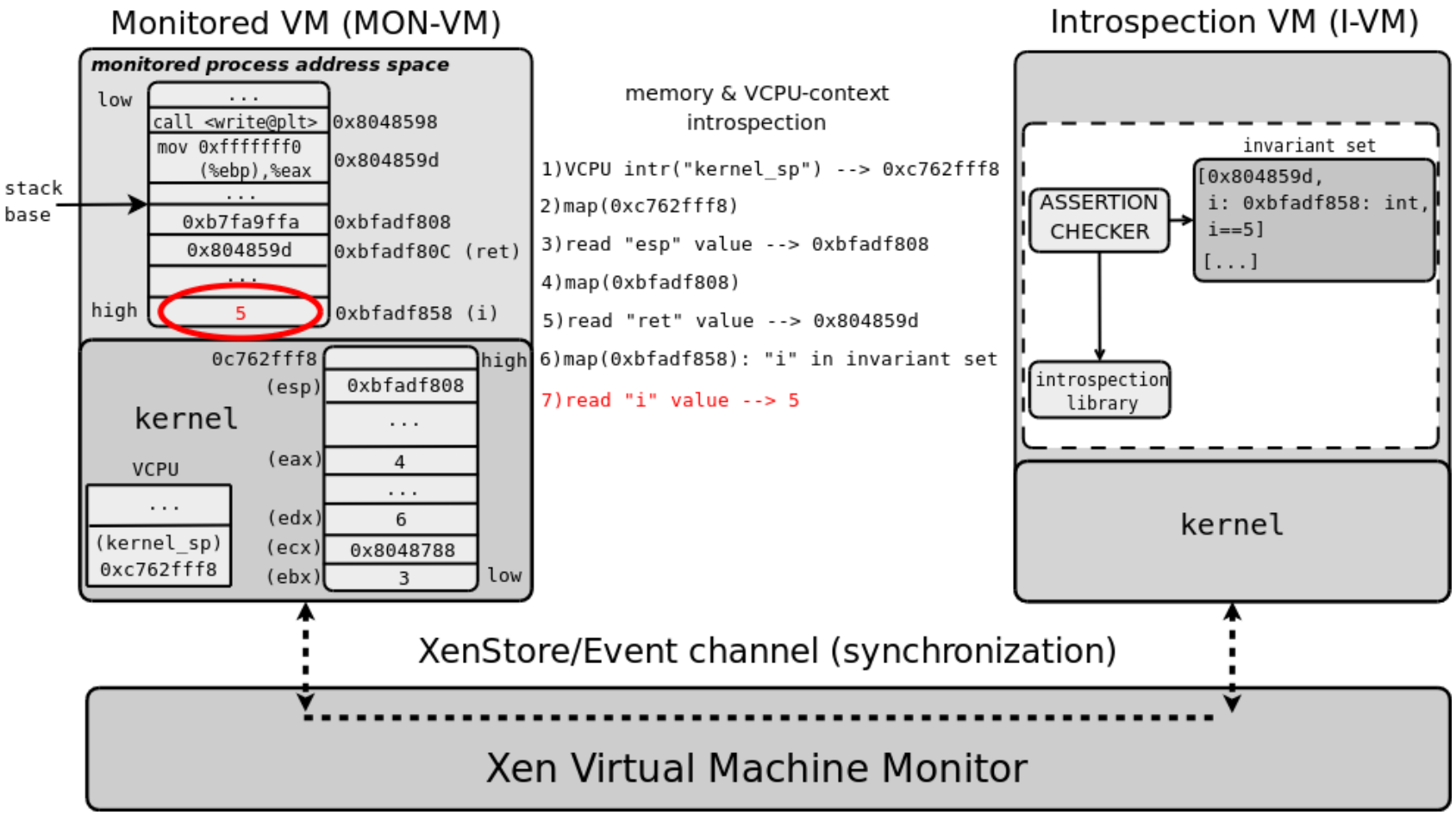
Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
...
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
high
5
0xbfadf858 (i)
stack base
0c762fff8
(esp)
0xbfadf808
high
kernel
...
(eax)
4
...
VCPU
...
(edx)
6
(kernel_sp)
0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
memory & VCPU-context introspection
1)VCPU intr("kernel_sp") --> 0xc762fff8
2)map(0xc762fff8)
3)read "esp" value --> 0xbfadf808
4)map(0xbfadf808)
5)read "ret" value --> 0x804859d
Introspection VM (I-VM)
invariant set
ASSERTION CHECKER
[0x804859d,
i: 0xbfadf858: int,
i==5]
[...]
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

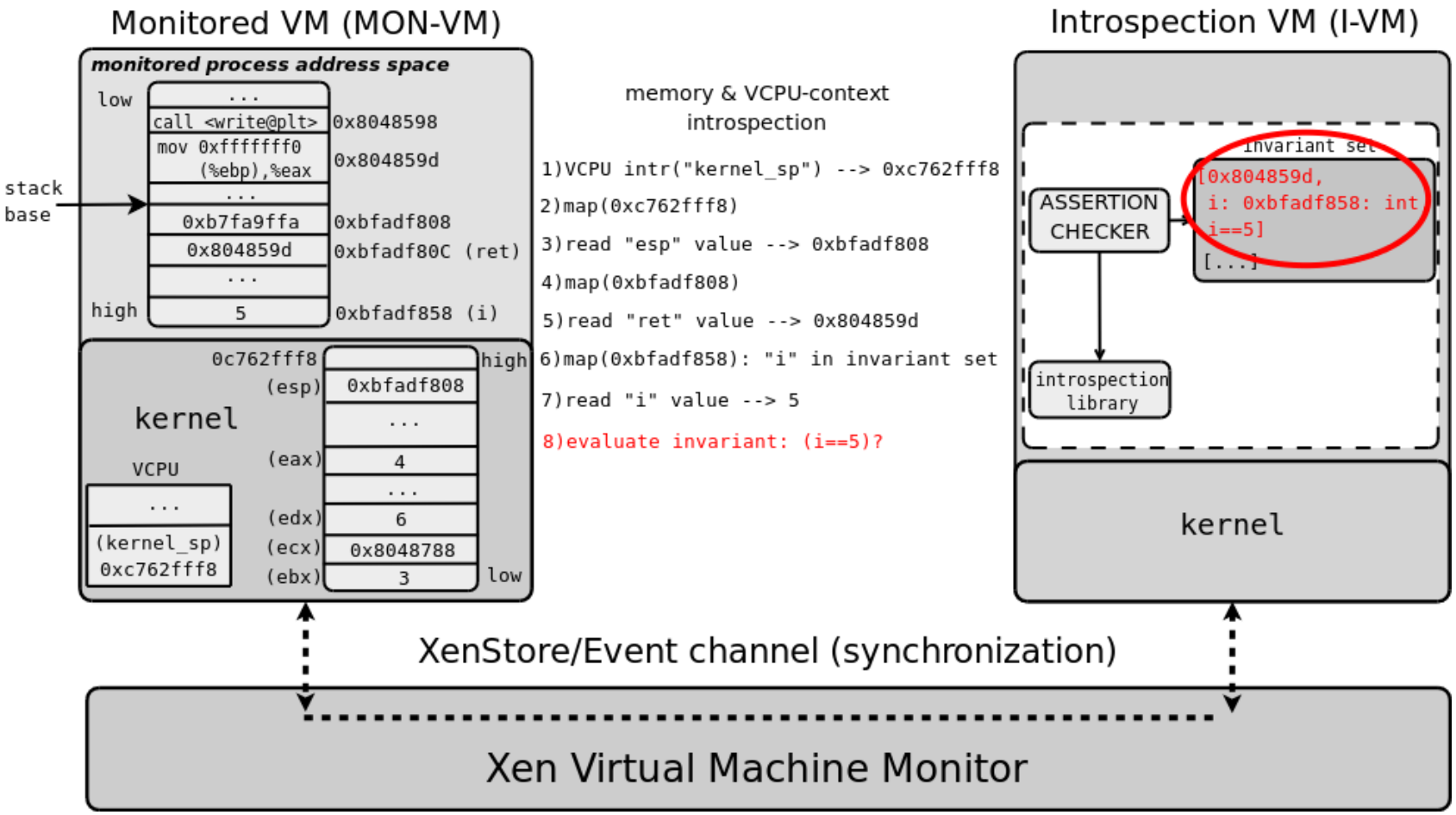
Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
...
stack base
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
...
high
5
0xbfadf858 (i)
0c762fff8
high
(esp)
0xbfadf808
kernel
...
(eax)
4
VCPU
...
...
(edx)
6
(kernel_sp)
0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
memory & VCPU-context introspection
1)VCPU intr("kernel_sp") --> 0xc762fff8
2)map(0xc762fff8)
3)read "esp" value --> 0xbfadf808
4)map(0xbfadf808)
5)read "ret" value --> 0x804859d
6)map(0xbfadf858): "i" in invariant set
Introspection VM (I-VM)
invariant set
ASSERTION CHECKER
[0x804859d, i: 0xbfadf858: int, i==5]
[...]
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
stack base
...
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
...
high
5
0xbfadf858 (i)
0c762fff8
(esp)
0xbfadf808
high
kernel
...
(eax)
4
...
VCPU
...
(edx)
6
(kernel_sp)
0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
memory & VCPU-context introspection
1)VCPU intr("kernel_sp") --> 0xc762fff8
2)map(0xc762fff8)
3)read "esp" value --> 0xbfadf808
4)map(0xbfadf808)
5)read "ret" value --> 0x804859d
6)map(0xbfadf858): "i" in invariant set
7)read "i" value --> 5
Introspection VM (I-VM)
invariant set
ASSERTION CHECKER
[0x804859d,
i: 0xbfadf858: int,
i==5]
[...]
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

Example


Monitored VM (MON-VM)
monitored process address space
low
...
call <write@plt>
0x8048598
mov 0xfffffff0 (%ebp),%eax
0x804859d
stack base
...
0xb7fa9ffa
0xbfadf808
0x804859d
0xbfadf80C (ret)
...
high
5
0xbfadf858 (i)
0c762fff8
(esp)
0xbfadf808
high
kernel
...
(eax)
4
VCPU
...
...
(edx)
6
(kernel_sp)
0xc762fff8
(ecx)
0x8048788
(ebx)
3
low
memory & VCPU-context introspection
1)VCPU intr("kernel_sp") --> 0xc762fff8
2)map(0xc762fff8)
3)read "esp" value --> 0xbfadf808
4)map(0xbfadf808)
5)read "ret" value --> 0x804859d
6)map(0xbfadf858): "i" in invariant set
7)read "i" value --> 5
8)evaluate invariant: (i==5)?
Introspection VM (I-VM)
invariant set
[0x804859d,
i: 0xbfadf858: int
i==5]
[...]
ASSERTION CHECKER
introspection library
kernel
XenStore/Event channel (synchronization)
Xen Virtual Machine Monitor

Performance Results

- The average time to map a page of the process into the Assertion Checker address space is about 50μsecs:
  - at least three pages (kernel stack, user stack, variable): 150μsecs.
- By exploiting a software TLB, if variables are stored in the same page, each access requires 20μsecs:
  - 60μsecs overhead for each evaluation.
- Taking into account the rate of system call invocations of `ghttpd` server, the average execution time overhead is at most 20%.

Results

- Semantics-driven approach to monitor program execution that exploits virtualization technology to:
  - access the process memory from a distinct VM;
  - evaluate an invariant for each system call.
- Non-control-data attacks.
- Isolation and robustness.
- Acceptable overhead.

Limitations and Future Works

- Currently focused on the run-time tool: retrieving variables and evaluate invariants.
- Automatic extraction of invariants from the application's source code of the monitored process.
- Retrieve the addresses of local variables:
  - e.g.: by keeping track of the value of the frame pointer and each variable's offset.
- Complete transparency: no need of hijacking system calls.

Questions?