rVMI

A New Paradigm for Full System Analysis

JONAS PFOH
SEBASTIAN VOGL

Overview

1. About us
2. Motivation
3. Architecture
4. Implementation
5. Demo
6. Summary

1. About us

1. About us

Dr. Jonas Pfoh

- Engineers at FireEye
- Work on malware detection
- Research Interests
  - Virtualization and Virtual Machine Introspection
  - Malware and exploitation techniques

Dr. Sebastian Vogl

2. Motivation

2. Motivation: What we did

rVMI: “Debugger on Steroids“

- Runs completely isolated form the target through **Virtualization**
- Full control over the target with **VMI**
- Leverages **memory forensiscs** to make the full state of the guest available to the analyst
- **Interactive** and **scriptable**

2. Motivation: Why we did it

Manual dynamic analysis

- Analyzing malware and exploits
- Gain a full understanding of a sample
- Tools
  1. Debuggers
  2. Sandboxes

2. Motivation: Why we did it

Debuggers

- Provide visibility and flexibility
- BUT: NEVER DESIGNED FOR MALICIOUS SOFTWARE
- Inherent attack vectors
    1. No isolation
    2. Rely on the OS to function and to obtain data
    3. Designed to trace a single process (or the kernel)

2. Motivation: Why we did it

Sandboxes

- Provide isolation and evasion resistance
- BUT: NEVER DESIGNED FOR INTERACTIVE ANALYSIS
- Manual analysis is difficult
    1. Limited Visibility
    2. Limited Control
    3. Limited Interface

2. Motivation: Why we did it

The current state of interactive dynamic analysis

- Neither debuggers nor sandboxes were designed for this
- We face a tradeoff between visibility/flexibility and isolation/evasion resistance
- We are not aware of any tool that combines these properties

→ We need a new tool that was designed for the job.

3. Architecture

3. Architecture: Goals

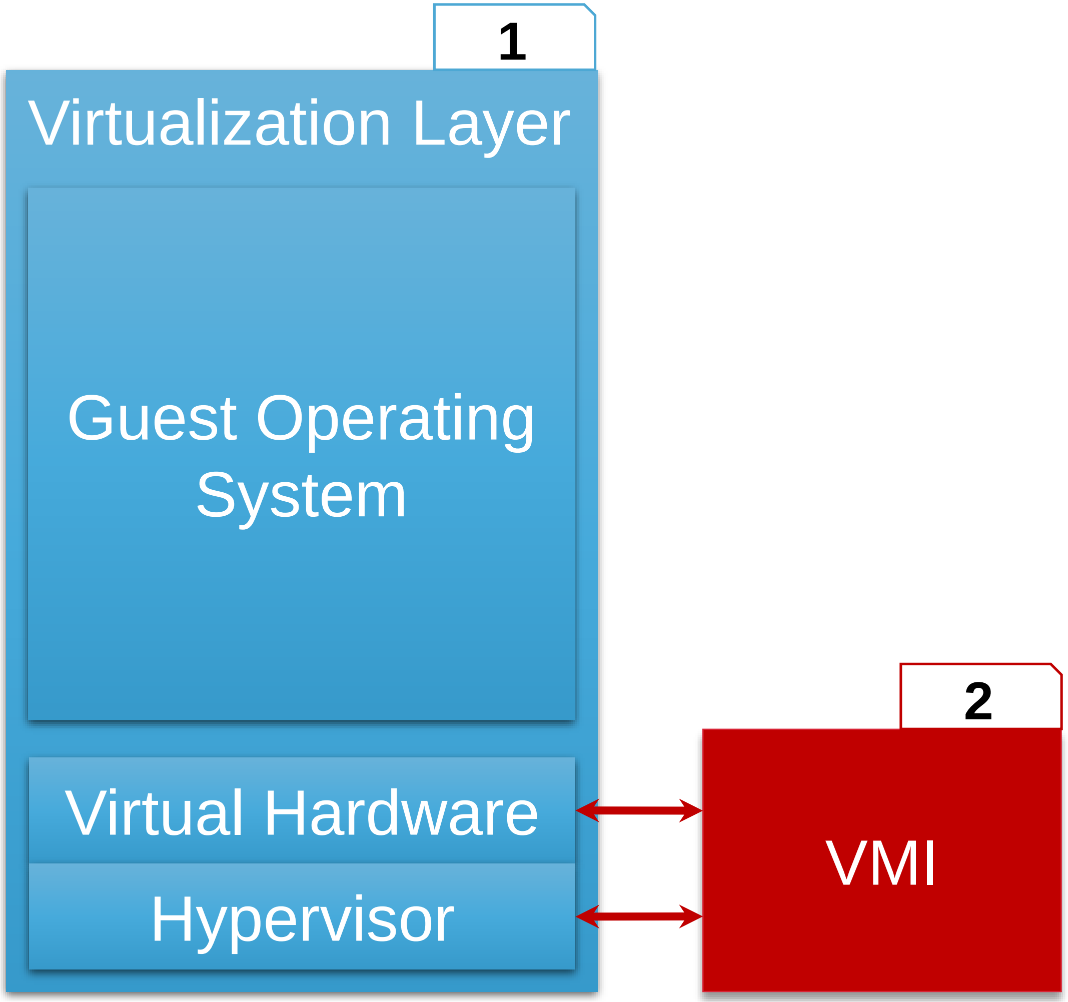
1. Resistant to evasion
2. Full system analysis
3. Interactive and scriptable

3. Architecture: Resistant to evasion

Building Block 1
Virtualization

- Isolate the target system from the analysis environment
- Increase evasion-resistance

3. Architecture: Full system analysis



Building Block II
VMI

- Provides full control of target system
- Allows access to entire state of the VM
- Exports this functionality through an API

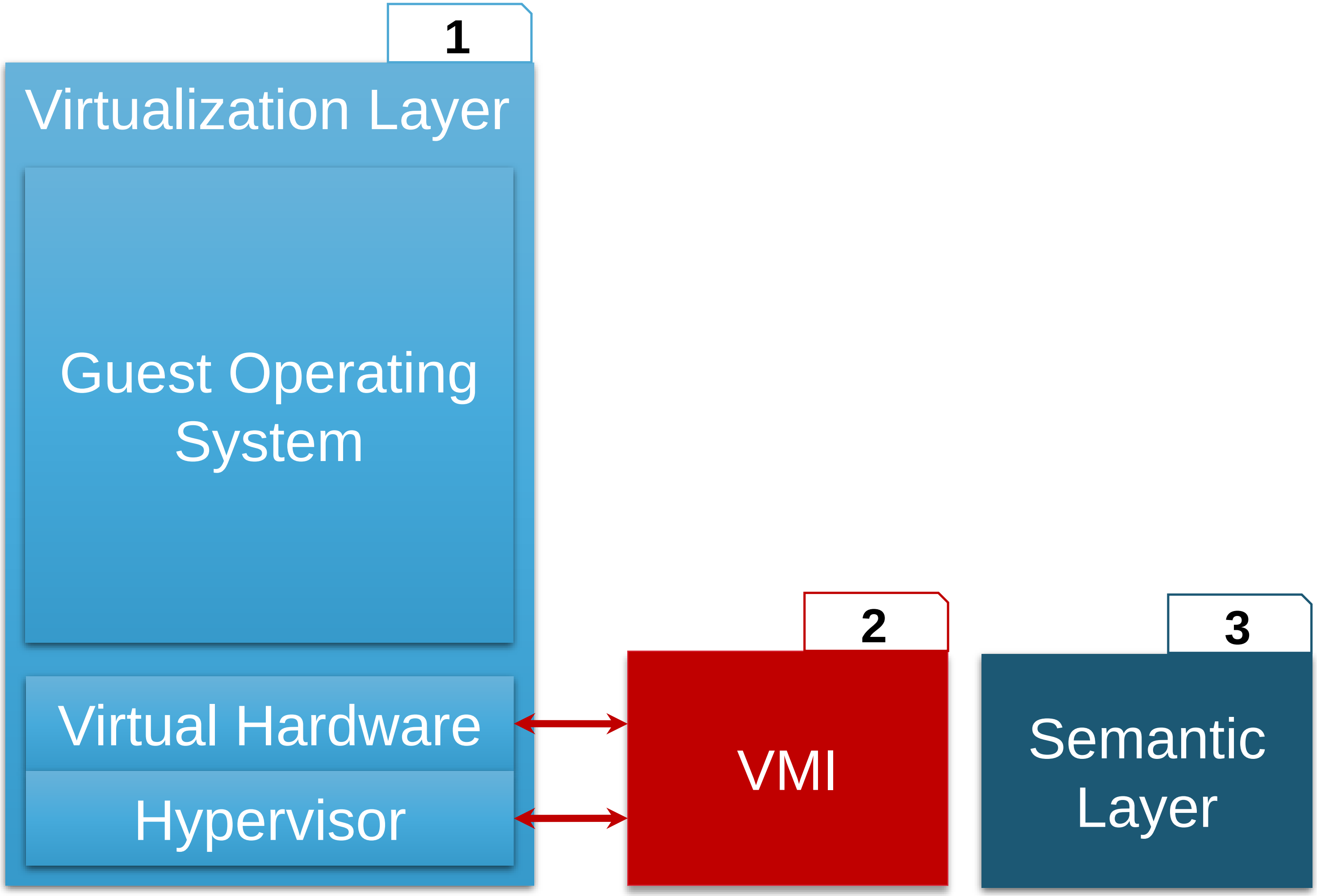
3. Architecture: The semantic gap

10100101001001000110100010111101010010110100010101110110
01101000101011101100010001011110101001001011100101100010
11110100010101110110000101010011110101001110010100001010
00001011101100010001011110100101001001000110100010111101
01110110000101010011110101100010101110110001000101111010
10010010111001011000101111010001010111011000010101001111
01010011100101000010100000101110110001000101111000011010
01110110000101010010100010010010100001001010001001000001
01111101010101111011110101100010101110110001000101111010
10010010111001011000101111010001010111011000010101001111

3. Architecture: The semantic gap

black hat USA 2017

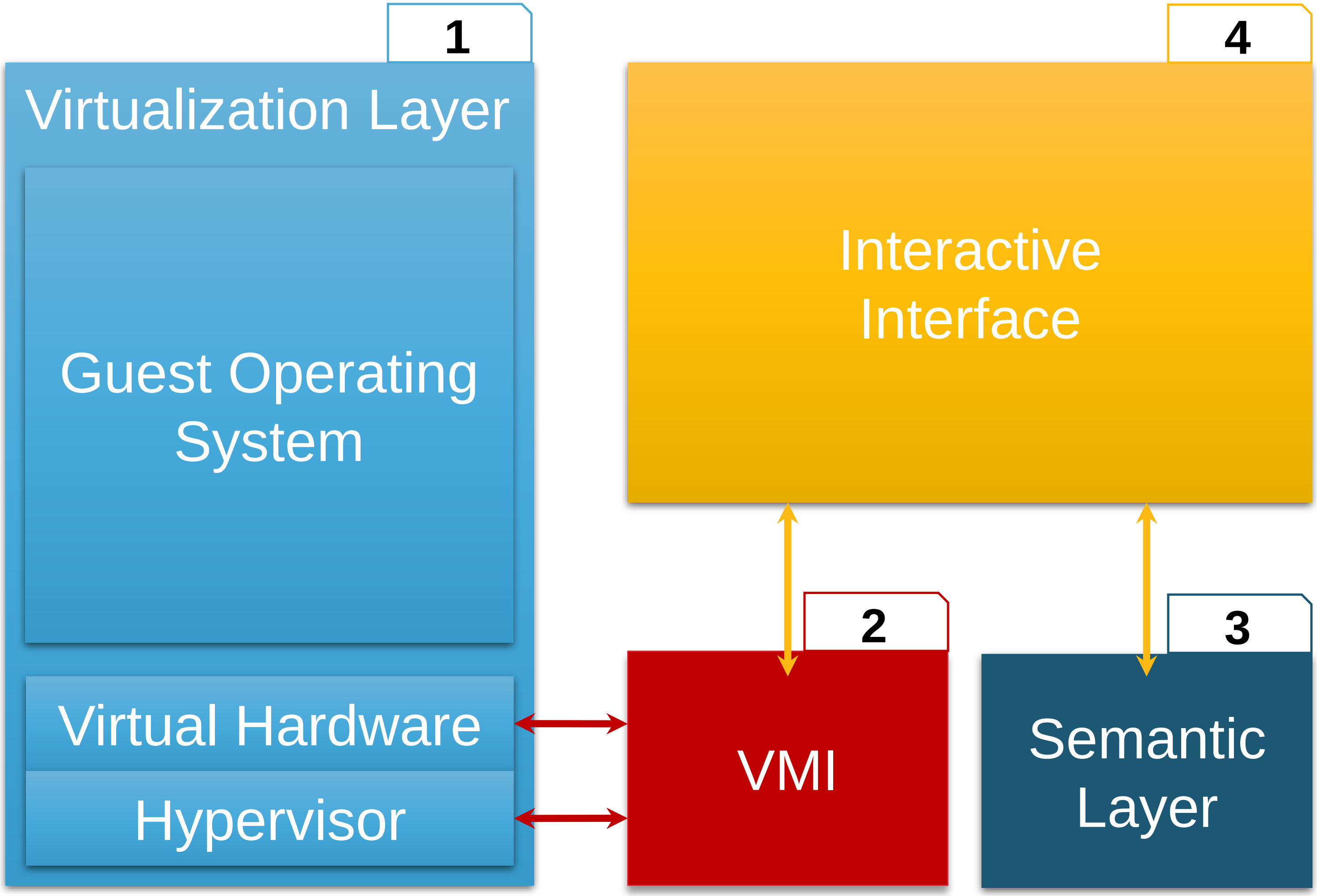
3. Architecture: Full system analysis



Building Block III
Semantic Layer

- Reconstruct the semantic knowledge of the guest OS
- Export this knowledge through an API

3. Architecture: Interactive and scriptable



Building Block IV
Interactive Interface

- Environment for the actual analysis
- Provides access to the VMI layer and the semantic layer
- Support for scripting

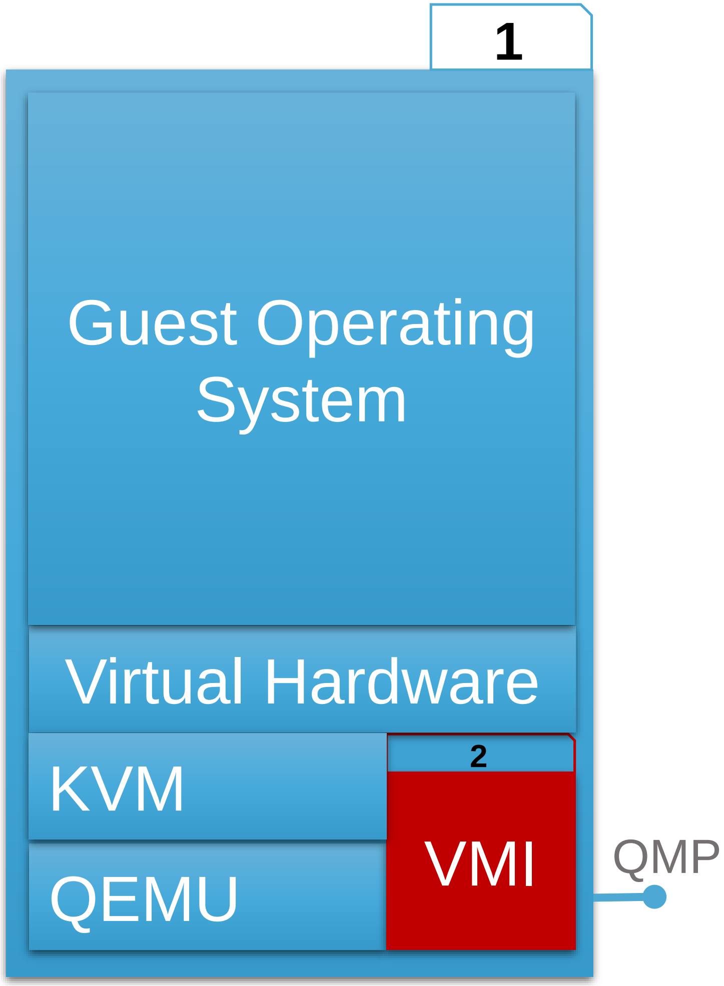
4. Implementation

4. Implemetation: Virtualization

Building Block I
Virtualization

- Chose KVM/QEMU as a basis as it is the natural choice for a debugger
- System does not depend on the hypervisor

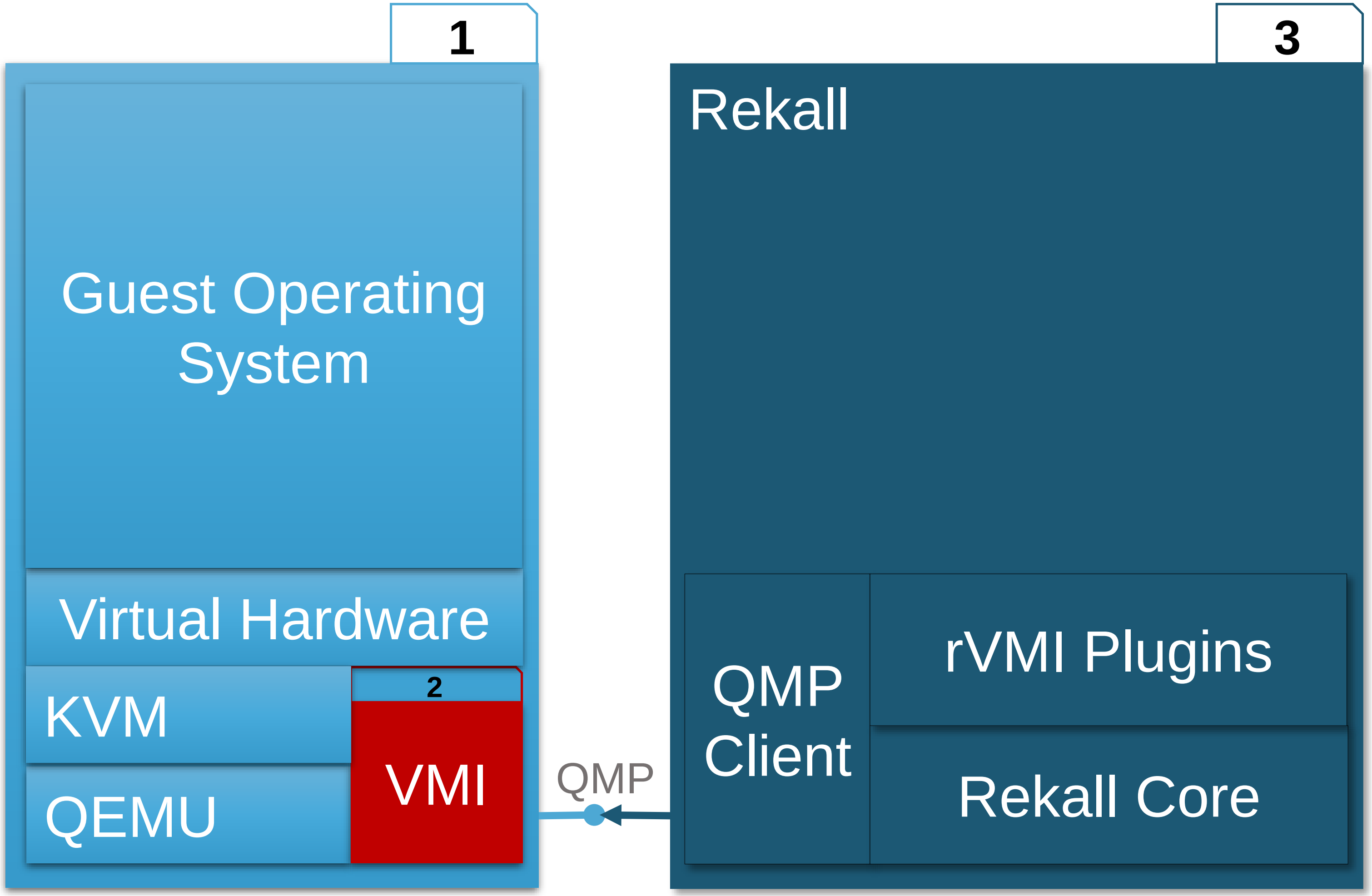
4. Implemetation: VMI



Building Block II
VMI

- Extended KVM/QEMU with a VMI interface
- Interface exposes the hardware features
- Exported this interface via QMP

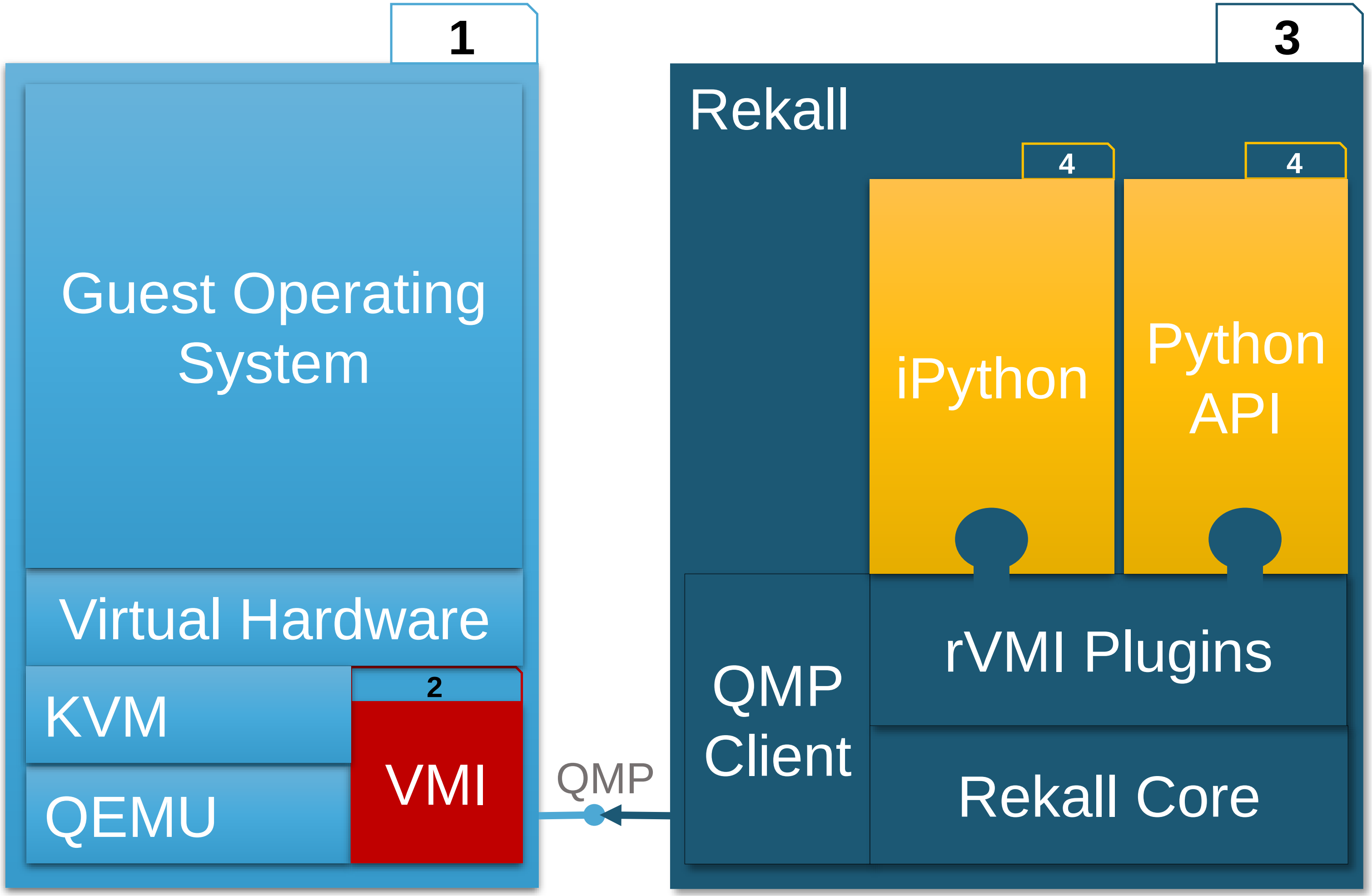
4. Implemetation: Semantic Layer



Building Block III
Semantic Layer

- Used Rekall as a basis
- Supports multiple OSs
- Can access OS structures in python
- Added a QMP address space to access the VM through VMI

4. Implemetation: Interface



Building Block IV
Interactive Interface

- Used the iPython shell of Rekall as basis for the debugger interface
- Added Rekall plugins for VMI commands
- Added rVMI events and exported an interface to these events

5. Demonstration

5. Demonstration

DEMO

6. Summary

Summary

- Interactive dynamic malware analysis is an important technique
- Currently no tools for the job
- We presented rVMI: Debugger on steroids
  - Isolation
  - Full control and full access
  - Interactive and scriptable

https://github.com/fireeye/rVMI

We are open to feedback and feature requests and hope for contributions.