Microsoft

Hypervisor-Enforced Kernel Integrity for Linux, powered by KVM

Linux Security Summit North America

Mickaël Salaün & Madhavan Venkataraman

2023-05-11

Protect Linux users against kernel exploits

Attackers can leverage security vulnerabilities using kernel exploits to get access to all users' data.

Kernel integrity

Security property guaranteeing critical parts of a virtualized Linux kernel to not be tampered by malicious code or data.

Problem

One kernel vulnerability is enough to bypass kernel integrity.

300+ Linux CVEs in 2022

including 8 code executions

Overview

Threat model

The attacker has arbitrary read and write access to the guest kernel e.g., thanks to exploited vulnerability by malicious

- User space process
- Network packet
- Block device

E.g., DirtyCOW (CVE-2016-5195)

Leverage virtualization

The main issue with kernel self-protection is that it is a self-protection.

Solution: rely on a higher privileged component: the hypervisor

State of the art: products and PoC

- grsecurity/PaX, OpenBSD
- Windows's Virtualization Based Security (i.e., HVCI, HyperGuard…)
- Samsung RKP, Huawei Hypervisor Execution Environment
- iOS Watchtower/KPP and RoRgn + KTRR
- Bitdefender's Hypervisor Memory Introspection
- Intel's Virtualization Based Hardening

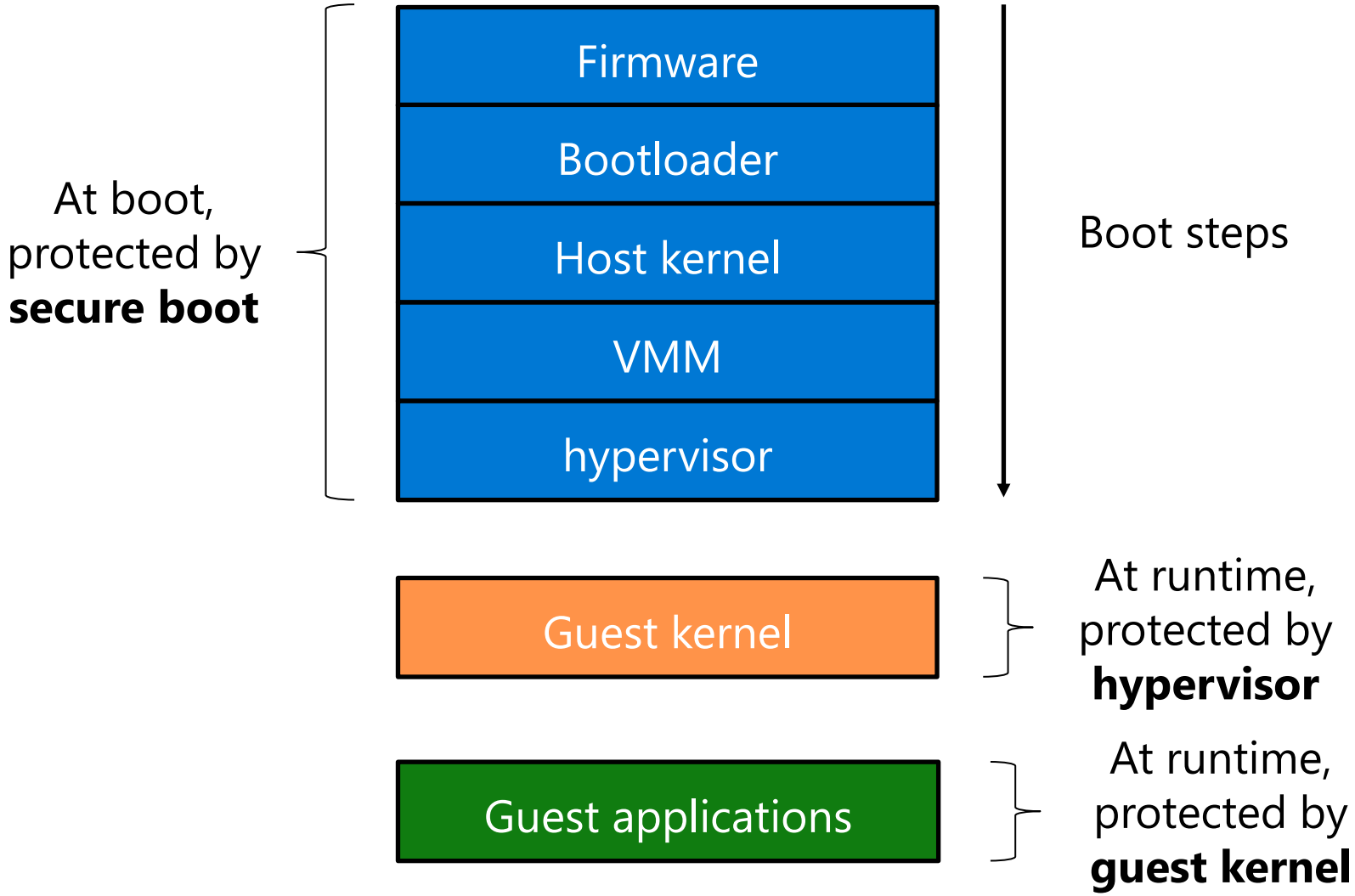
State of the art: proposed Linux patches

- KVMI: VM introspection
- Paravirtualized Control Register pinning
- Hypervisor Based Integrity

KVM

- Hypervisor part in the host kernel
- Driver part in the guest kernel
- Leverage Linux process management mechanisms for VM (e.g., scheduling, resource limitation)
- Configured and controlled by a Virtual Machine Monitor (VMM) e.g., Qemu, Cloud Hypervisor

Chain of trust



Prerequisites

Usability:

- Users manage their own kernels
- This feature needs to be simple to use and standalone: almost no configuration

Security:

- Trusted and secure VMM/hypervisor

Security policies

Improve Linux kernel hardening:

- Enforce critical CPU register pining: CR0.WP, CR4.SMEP, CR4.SMAP, CR4.UMIP, CR4.FSGSBASE, CR4.CET
- Enforce read-only kernel data (e.g., syscall table, certificates, keys, security configuration)
- Enforce a global *execute XOR write* kernel memory policy

Common guest kernel API

Design

- The guest VM configures itself with a hypervisor-agnostic API
- The hypervisor manages enforcements
- The VMM could get attack signals

Guest API

Normalized common layer that can be used by any hypervisor to receive guest requests:

- Map kernel memory pages with required permissions or attributes
- Hide hypervisor implementation details (e.g., hypercalls)
- Shared test suites

Guest API

```
#define HEKI_ATTR_MEM_NOWRITE		(1ULL << 0)
#define HEKI_ATTR_MEM_EXEC		(1ULL << 1)

struct heki_va_range {
	void *va_start;
	void *va_end;
	u64 attributes;
};

struct heki_hypervisor {
	int (*protect_ranges)(struct heki_pa_range *ranges,
			      int num_ranges);
	int (*lock_crs)(void);
};
```

Guest boot

kernel_init:

1. heki_early_init: Configure the protections (i.e., kernel sections)
2. mark_readonly: Enforce MMU protections
3. heki_late_init: Enforce EPT and CR protections
4. Launch init process

KVM implementation

Security guarantees

Protect control register modifications and enforce kernel memory restrictions according to the guest requests: 2 new hypercalls.

It can only add more restrictions, not remove them.

CR-pinning hypercall

Enforce a bitmask to guard against locked features (e.g SMEP).

```
kvm_hypercall2(KVM_HC_LOCK_CR_UPDATE,
                0, // control register
                X86_CR0_WP); // flag to pin
```

Memory protection hypercall

Track EPT page faults related to these pages

- Configure VMCS
- Log violation attempts
- Generate synthetic page fault for the guest

```
kvm_hypercall3(KVM_HC_LOCK_MEM_PAGE_RANGES,
                __pa(ranges), // physical address ranges
                size, // size of the array
                0); // options
```

SLAT/TDP

Part of hardware virtualization (e.g., Intel's VT-x)

Second Layer Address Translation or Two Dimensional Paging:

- Intel's EPT
- AMD's RVI/NPT

Enable to manage VM memory, and add a second complementary layer of permissions, only controlled by the hypervisor.

Kernel execution

3 memory page permissions:

- Read
- Write
- Execute

Issue: Too coarse, no efficient way to take different decision based on the execution mode.

MBEC

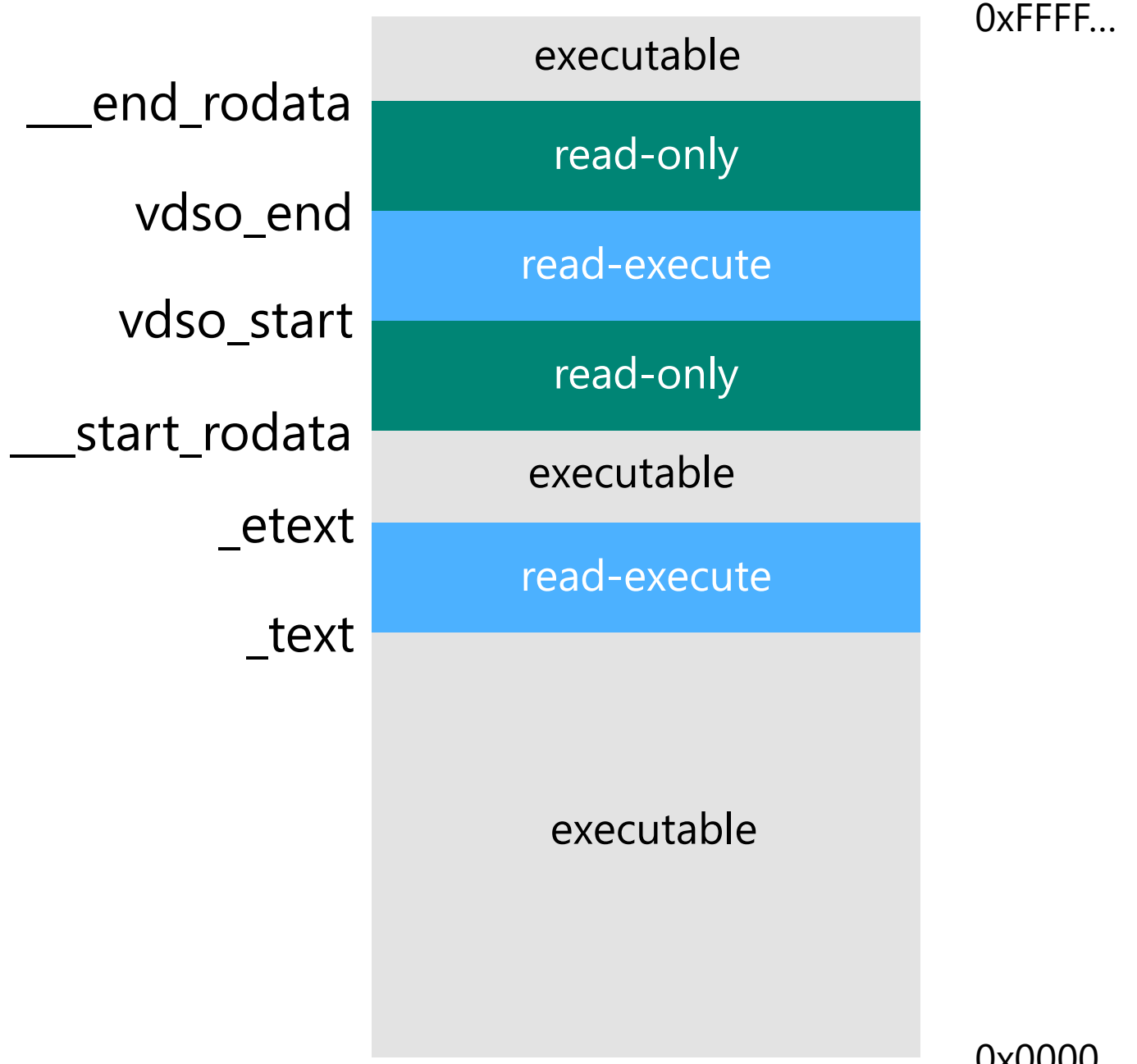
Mode Based Execution Control

Split the execution permission into:

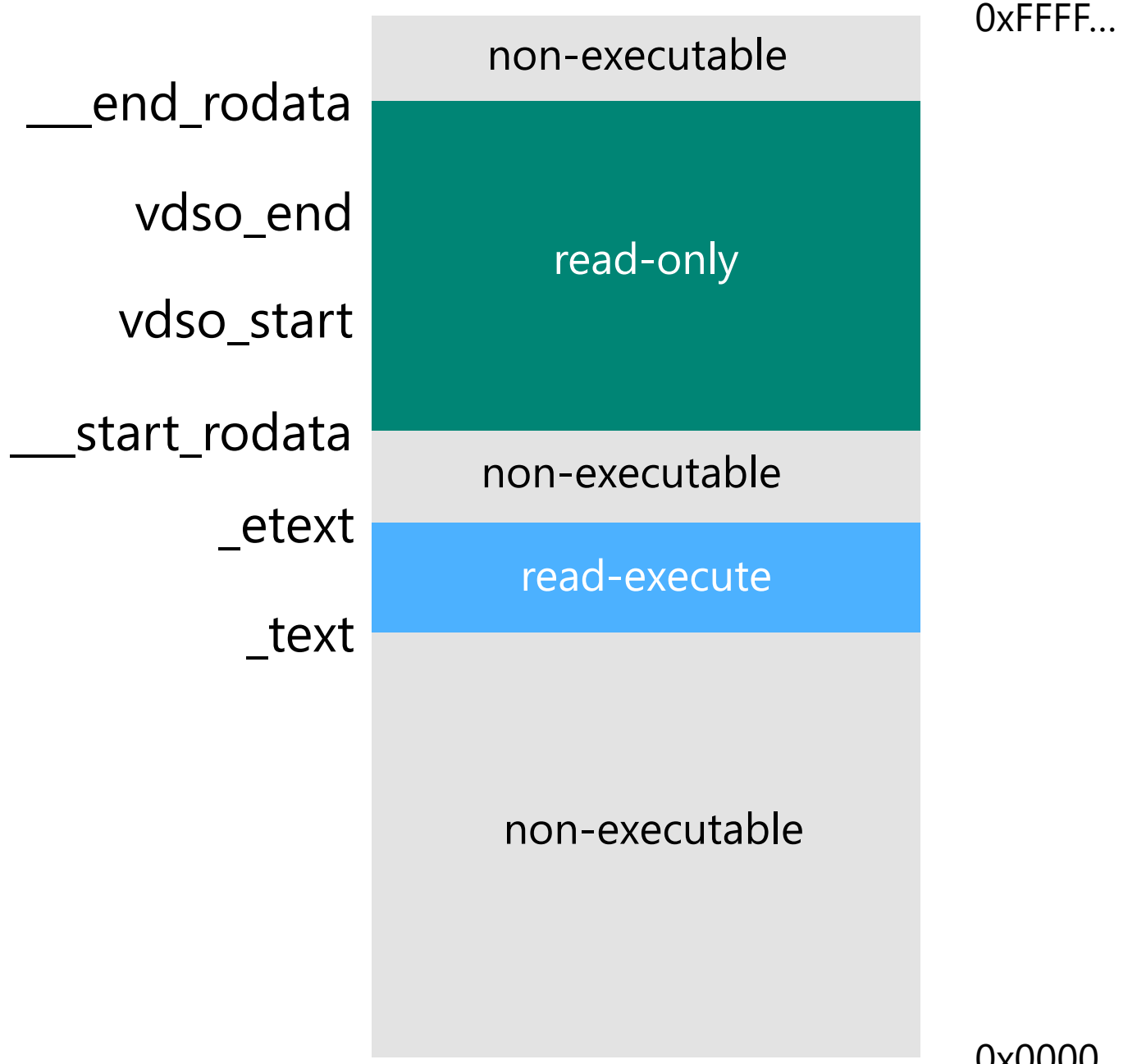
- Kernel mode execution
- User mode execution

Heki leverages MBEC (which is a requirement) to enforce a deny-by-default kernel memory execution policy.

Kernel memory permissions without MBEC



Kernel memory permissions with MBEC



Demo: kernel memory protections

```
long heki_test_exec_data(long);
void _test_exec_data_end(void);

/* Used to test ROP execution against the .rodata section. */
/* clang-format off */
asm(
".pushsection .rodata;" // NOT .text section
".global heki_test_exec_data;"
".type heki_test_exec_data, @function;"
"heki_test_exec_data:"
ASM_ENDBR
"movq %rdi, %rax;"
"inc %rax;"
ASM_RET
".size heki_test_exec_data, .-heki_test_exec_data;"
"_test_exec_data_end:"
".popsection");
/* clang-format on */
```

97,1 29%

```
user@heki-host$ git diff
diff --git a/virt/heki/heki.c b/virt/heki/heki.c
index 361e7734e950..df85aa9758b3 100644
--- a/virt/heki/heki.c
+++ b/virt/heki/heki.c
@@ -52,10 +52,10 @@ struct heki_pa_range *heki_alloc_pa_ranges(struct heki_vr,
                 * Leaks addresses, should only be kept for development.
                 */
                attributes = pa_range->attributes;
-               pr_warn("Configuring GFN 0x%llx-0x%llx with %s\n",
+               pr_warn("Configuring GFN 0x%llx-0x%llx with %s%s\n",
                        pa_range->gfn_start, pa_range->gfn_end,
-                       (attributes & HEKI_ATTR_MEM_NOWRITE) ? "[nowrite]" :
-                                                              "");
+                       (attributes & HEKI_ATTR_MEM_NOWRITE) ? "[nowrite]" :",
+                       (attributes & HEKI_ATTR_MEM_EXEC) ? "[exec]" : "");
        }

        return pa_ranges;
user@heki-host$
```

Demo: control-register pinning (SMEP)

```
user@heki-host$
```

```
static void heki_test_cr_disable_smep(struct kunit *test)
{
        unsigned long cr4;

        /* SMEP should be initially enabled. */
        KUNIT_ASSERT_TRUE(test, __read_cr4() & X86_CR4_SMEP);

        kunit_warn(test,
                   "Starting control register pinning tests with SMEP check\n")
;

        /*
         * Trying to disable SMEP, bypassing kernel self-protection by not
         * using cr4_clear_bits(X86_CR4_SMEP).
         */
        cr4 = __read_cr4() & ~X86_CR4_SMEP;
        asm volatile("mov %0,%%cr4" : "+r"(cr4) : : "memory");

        /* SMEP should still be enabled. */
        KUNIT_ASSERT_TRUE(test, __read_cr4() & X86_CR4_SMEP);
                                                    124,1-8       36%
```

Current limitations

- Statically enforced permissions: no kernel module nor dynamic new kernel code (e.g., tracepoints, eBPF JIT)
- ROP protection is out of scope: need to rely on CFI

Future work #1

Securely handle dynamic code execution allowed by an external entity doing the code authentication:

- The VMM, or
- A dedicated bodyguard VM (cf. VBS's VTL)

Freeing memory (e.g., kernel module) requires to get back to safe default permissions.

Future work #2

Improve kernel self-protection mechanism and duplicate them at a higher level

Improve hardening:

- Extend register protection (e.g., MSRs)
- Support Hypervisor-managed Linear Address Translation
- Support eXecute-Only-Memory (XOM)
- Protect KVM’s host

Future work #3

Monitor attacks:

- New interface to log attack attempts
- React to an attack

Usability:

- Support nested VMs
- Support more architectures

Wrap up

Heki is a defense-in-depth mechanism leveraging hardware virtualization, especially MBEC.

The Linux RFC defines a common API layer across hypervisors.

This API is used in a freely available proof-of-concept for KVM, that doesn’t require VMM changes.

Test and contribute!

We’re looking for contributions!

- New hypervisors support
- New architecture support
- Improved guest kernels support
- VMM enhancements

https://github.com/heki-linux