Google Cloud

Hypervisor Based Integrity: Protect Guest Kernel in Cloud

Ning Yang (ningyang@google.com)

Forrest Yu (yuanyu@google.com)

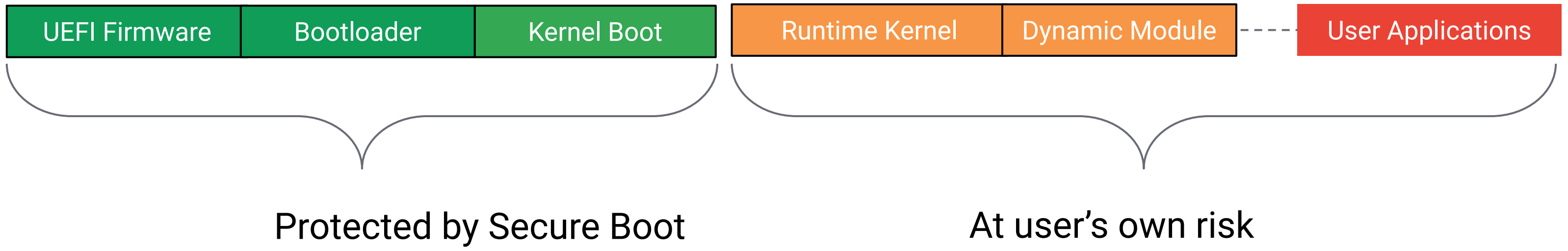
KVM Forum 2020

Disclaimer

- This is not a statement of direction or planned investment by Google.
- We are exploring this as a possible security mechanism and seeking feedback from the community.

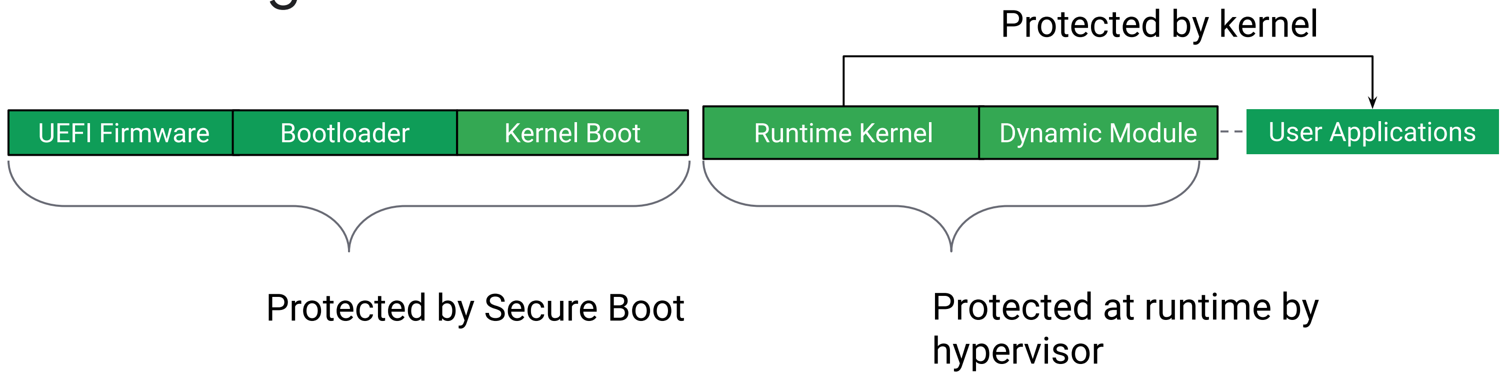
Background

- Google Cloud Offers Shielded VM with Secure Boot
  - It provides integrity check against firmware, bootloader and kernel during boot
  - most kernel modules are protected by module signing.
- At run time, attacks against the kernel could still happen:
  - If the kernel is compromised, all existing guest kernel security protection mechanisms cannot be trusted



Background

- The goal is to protect the kernel at runtime, so it can be trusted and the user applications are safe
- The protection put in place cannot be turned off from the guest



Threat Model

Attacker capabilities:

- Arbitrary read/write primitives in the Guest: they can read any kernel memory address, and write any value to any kernel memory address
- Cannot break into host/hypervisor: based on the strong vm isolation and assume the hypervisor does not have other security vulnerabilities.

We base this model on exploits found in the wild. For additional information, see PZ report on the Android binder exploit.

Protect the guest kernel at runtime

- No unintended modifications of the kernel .text and .rodata segments
- No code execution from other parts of kernel space
  - e.g. data segment should be R/W but not X
  - All executable memory should be non-writable
- No unintended modification of key kernel data structures e.g:
  - system call table
  - control registers(CR0, CR4) or important MSR(like sysenter)
  - IDT/GDT
  - page tables

Why hypervisor as another security layer?

There is always risk protecting the guest from within the guest:

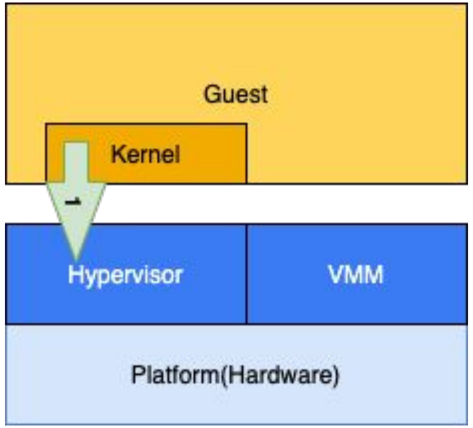
- Rootkits can potentially gain the highest privilege and access all of guest memory including the kernel
- Anything we put inside the guest kernel could be at risk but if we keep the kernel safe at run time, all the security gates implemented in the kernel will become more robust

Why hypervisor as another security layer?

- Hypervisor can control R/W/X for page-aligned guest memory regions
  - Two-Dimensional paging (EPT for Intel, NPT for AMD)
- Hypervisor can protect unsafe modification of CRs and MSRs
- Hypervisor can protect key kernel page tables:
  - HYPERVISOR-MANAGED LINEAR ADDRESS TRANSLATION from Intel.
- Cloud industry already has all the guest VMs running under the control of a hypervisor

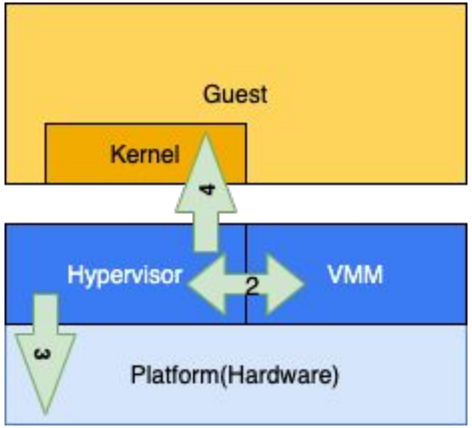
Overall Plan - Boot Time: Guest

- New Guest Security kernel module:
  - This module would be loaded at boot time
  - It should be signed and protected by Secure Boot so it can be trusted at boot time
  - Check whether the hypervisor supports integrity protection
  - Identify the kernel code/data segments and addresses of key kernel data structures
  - Send all the information to the underlying hypervisor and wait for the acknowledgement



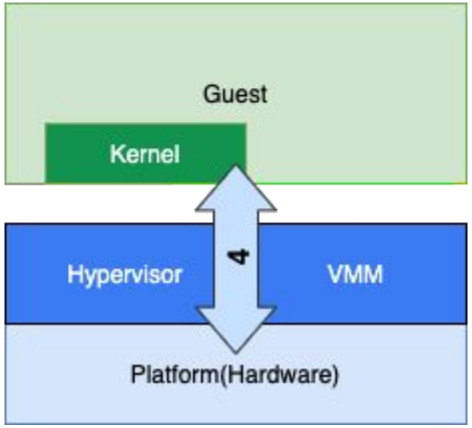
Overall Plan - Boot Time: Hypervisor

- Hypervisor(KVM) + VMM:
  - Decode the memory segment information sent from the guest module
  - Modify the second level address translation table to set correct R/W/X bits for the specified memory regions
  - Configure the MSR/CR exits
  - ACK the kernel module request and let it continue to boot



Overall Plan - Run Time

- Hypervisor(KVM) + VMM:
  - Handle EPT/NPT access permission violation VM exit, user can configure one of the following:
    - Kill the running VM and dump memory for analysis
    - Log critical event
  - Prohibit unintended CR/MSR modification.
  - What about dynamic kernel modules?
    - Guest kernel uses API exposed by the security module to allocate a R/W memory region
    - Copies the binary into it
    - Once attestation is done, call API to change .text to R/X to make sure it never gets changed again



Performance Impact

- Memory Access: minimal
  - Support through hardware: Intel EPT, AMD NPT
  - If no violation happens, there should be no impact at all
  - If a prohibited action happens, paying some extra cost on VM exit to determine if the operation should be permitted
- CR/MSR: minimal
  - We don’t expect lots of modification in the typical kernel run time
- Page Table Translate (HLAT):
  - Need Hardware support but currently limited to Intel Alder Lake

Technical challenges

- How do we distinguish legitimate kernel modifications?
  - Static keys
  - Kernel probes
  - Kexec activity
  - Dynamic kernel patching e.g. kpatch
  - Alternative macro
  - And more that we might not know about
- Need to disable those in the guest image or whitelist these operations
- When set memory permissions, not all segments are nicely aligned

Example

- We map entire kernel code segment as R/X.
  - Guest OS: Debian 10
- Got EPT access violation right away:

```
VCPU MMIO exit: RIP: 0xFFFFFFFFA3835106; CS: Base 0x
```

- We dump the call stack and analyze it:

```
ffffffffb9a2d2a0 T text_poke
ffffffffb9efe570 T netif_receive_skb_list
ffffffffbaac8500 d netstamp_work
```

- Comes from text_poke:

```
text_poke - Update instructions on a live kernel
```

- After removal of offending module, the guest runs smoothly without any further violations

Changes needed for KVM/QEMU

- Common interface to turn on protection:
  - Currently we are using a specific MSR to configure the protection
  - A new hypercall would be ideal and show it is for VM only
  - Once people agree on the interface, all hypervisors and VMMs can implement the API to provide the same protection regardless of where the VM lands

Changes needed for KVM/QEMU

- First RFC: add hypercall for KVM_HC_UCALL
  - link: sends a message from guest to VMM
  - Currently all hypercalls are handled inside KVM
  - We want a common hypercall interface to pass control back to the VMM
- Why have the VMM handle the requests:
  - It sets up the guest memory mapping and should also control/have knowledge of all permission settings
  - Simplify support for live migration
  - Keep KVM simple as it just needs to provide operations to change the protection on EPTs

Changes needed for KVM/QEMU

- Follow up RFC: define common interface for turning on protection
- Define a message from guest to VMM with structures like:

```
enum class Opcode : uint32 {
  kUnset,
  kSetMemoryProtection,
  ...
};

enum class MemoryPermission : uint32 {
  kReadAndExecute,
  kReadAndWrite,
}
```

```
struct HbiRequest {
  uint32 version;
  Opcode op_code;
  union {
    struct {
      // Physical page number for the start of the protected
memory region.
      uint64 physical_page_number;
      // Length of the protected memory region in page size.
      uint64 num_of_pages;
      // permission for this memory region.
      MemoryPermission permission;
    } set_memory_protection_request;
    ...
  };
  ReturnCode return_code;
};
```

Changes needed for KVM/QEMU

- Extend KVM_SET_USER_MEMORY_REGION
  - The current call only supports setting the memory as R/X
  - We need support for R/W but not X
- Expose VM Exit on CR Modification
  - CR0: PE, WP, PG, CD, NW, etc
  - CR4: SMEP, SMAP, PAE, PGE, etc
- Support for Intel's Hypervisor-Managed Linear Address Translation (HLAT)

For the Future

- Guest security module could expose APIs for the kernel to consume
- Guest kernel code can call the API to map key data regions as read-only at boot time and the hypervisor can guarantee they never change at runtime
- Current kernel security or driver modules can use this to protect themselves

Other security considerations

Why can we trust Secure boot in Google Cloud?

- Guest firmware(UEFI) is immutable
- Integrity checks for secure boot done in VMM process not in guest
- Secure boot variables stored in a remote service
  - no way to access/modify persistent memory used for variables

No potential SMM attacks

- SMM not supported for Google Cloud VMs

Return-oriented programming(ROP) attacks

- System still subject to ROP attacks to modify kernel page tables
- Attacker manages to change page table and point it to a different executable memory region which is not protected by hypervisor

Summary

- Proposed HV enforced protection for guest memory and system registers
- Proposed PV interface for protection enablement
- Secure by default, this hardening extend secure boot and provides runtime protection for kernel integrity

Appendix: Security Analysis

- Hypervisor Based Integrity(HBI) blocks can many attacks that lead to malicious code running in the kernel:
  - Non-executable bit in NPT prevents addition of new code pages.
  - Non-writable bit in NPT prevents modification of existing code pages.
- HBI raises the cost of attacks, and pushes attackers to pursue data only or ROP based attacks.
- HBI is a defense in depth measure, and should be enabled with additional protection mechanisms, such as the "Kernel self-protection".
- Future advances in kernel security, such as Kernel Control Flow Integrity and Intel CET, may mitigate the ROP attacks.