Hardening Linux guest kernel for CC

Elena Reshetova, Sathyanarayanan Kuppuswamy, Alexander Shishkin

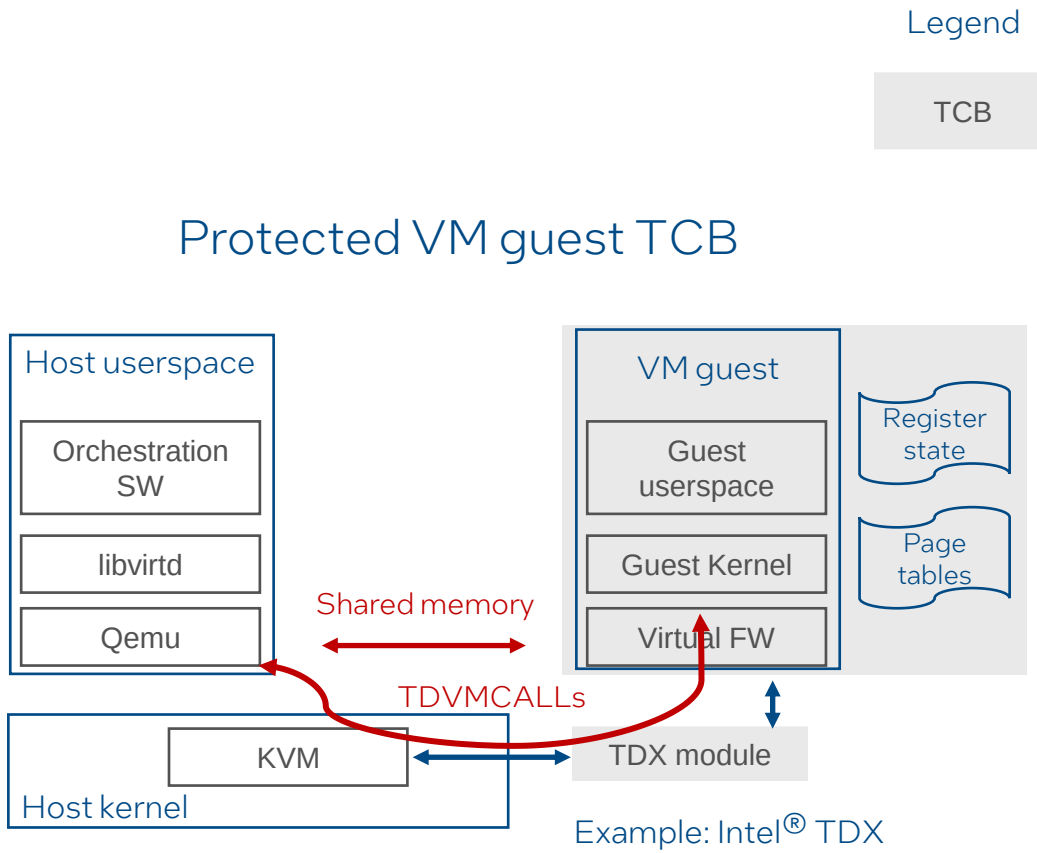
Confidential Computing MC, Linux Plumbers

13.9.2022

Why to harden?

VMM <-> Guest runtime interaction
- Paravirt (MSRs, PortIO, MMIO, CPUID, ...)
  - TDX specific hypercall (TDVMCALL)
- DMA shared memory regions (i.e. virtIO)

- Big and distributed attack surface*
  - Applies to all Confidential Cloud Computing (CCC) technologies
  - Input handling complexity varies (simple bit reads ←→ complex struct parsing)
  - Surrounding code has not been specifically written to withstand a maliciously crafted input
  - A single bug can be enough to gain a read or write primitive inside a guest kernel



*5.11 + desktop config: 26371 distinct guest kernel locations (90% in drivers)

Device filter

Problem: Drivers are 90% of threat attack surface (see slide 2)

- Device drivers are not written with CC threat model in mind
- Too many to be able to harden/audit

Solution: Device filter: disable in runtime all devices except a pre-defined allow list

- Currently allow list for buses: pci, acpi, platform
- Current list of PCI authorized devices for TDX guest:
  - virtio-block, virtio-net, virtio-console, virtio-9p and virtio-vsock
- Command line option to extend allow list

Alternatives?

- Minimal CC guest kernel config?
  - Doesn't fit vendor model of single distro config
  - Needs constant maintenance to make sure all devices are disabled
  - Not clear if it can disable all drivers, including platform drivers

Upstream feedback (Greg):

1. Should be done in driver core
2. Should take into account usb/thunderbolt cases
3. Allow list is better managed by userspace/initrd

- Current status:
  - 4 patches that address 1 & 2:
    - https://github.com/intel/tdx/commits/guest-filter-upstream
  - Aspect 3 still needs patch(es) created
- Opens:
  - Support for varied PCI IDs (NVMe direct assignment, ..)
    - Based on class: PCI_CLASS_STORAGE_EXPRESS ?
    - Would class-based cover all generic cases?
  - ACPI shared memory regions allocation
    - Many ACPI drivers don't do ioremap calls by themselves, but use (via AML methods) OperationRegions declared in their AML objects
    - Changing ACPI core to automatically share all touched regions opens attack surface too much

Paravirt-based communication interfaces

Problem: these interfaces can be used as attack vector from host/VMM

- Device filter should cover most of attack surface, but
- Many drivers will consume host input already their init functions
  - 5.15-based TDX kernel: 198 unique __init functions, 5198 unique code locations
    - **pci config space:** 83 unique __init functions, 772 unique code locations
    - **msrs & apic:** 72 unique __init functions , 91 unique code locations
    - rest is MMIO & port IO
  - Data for 5.15-rc1:
    - https://raw.githubusercontent.com/intel/ccc-linux-guest-hardening/master/bkc/audit/sample_output/5.15-rc1/smatch_warns_5.15_tdx_allyesconfig_driver_analysis

Will cover now solutions for:

- MMIO & shared memory
- PCI config space
- Port IO

- See backup slides for details:
  - MSRs
  - CPUIDs
  - KVM hypercalls and KVM CPUIDs

MMIO & shared memory

MMIO

- MMIO from userspace is disabled
- Kernel MMIO: opt-in shared with VMM using ioremap_driver_hardened()
- Automatically for authorized pci drivers & MSI mailbox
- Code:
  - https://github.com/intel/tdx/commit/90bd3f
  - https://github.com/intel/tdx/commit/80888b76
  - https://github.com/intel/tdx/commit/a790083
  - https://github.com/intel/tdx/commit/16097a5
  - https://github.com/intel/tdx/commit/16b90c1
- Opens:
  - ACPI OperationRegion support is still open

Shared memory: VirtIO

- Only split virtqueue without indirect descriptor support
- No support for virtio-pci-legacy mode and virtio-mmio
- Payload data is untrusted and must be verified separately
- Code:
  - https://github.com/intel/tdx/commit/c39c252d6
  - https://github.com/intel/tdx/commit/3377879

PCI config space

- Only through CF8: MCFG config space is disabled
- PCI config space access is blocked from non-allowed devices
  - 1 patch: github.com/intel/tdx/commit/1507340

Opens:

- Hotplug support

Port IO & Port IO filter

- Port IO filter: small allow list of ports in #VE handler
  - Only early and normal port IO
  - No filtering in decompressed mode
  - RTC, PCI config space, ACPI ports
  - Code: https://github.com/intel/tdx/commit/58ca3fe
  - Port IO from userspace is not supported

Opens:

- Having a hardcoded port list in kernel is likely not good
- Some ports might need to be open based on ACPI table configuration

```
239         switch (port) {
240         /* MC146818 RTC */
241         case 0x70 ... 0x71:
242         /* i8237A DMA controller */
243         case 0x80 ... 0x8f:
244         /* PCI */
245         case 0xcd8 ... 0xcdf:
246         case 0xcf8 ... 0xcff:
247                 return true;
248         /* PCIE hotplug device state for Q35 machine type */
249         case 0xcc4:
250         case 0xcc8:
251                 return true;
252         /* ACPI ports list:
253          * 0600-0603 : ACPI PM1a_EVT_BLK
254          * 0604-0605 : ACPI PM1a_CNT_BLK
255          * 0608-060b : ACPI PM_TMR
256          * 0620-062f : ACPI GPE0_BLK
257          */
258         case 0x600 ... 0x62f:
259                 return true;
```

ACPI & ACPI filter

Problem: ACPI tables can be used as an alternative attack vector

- In TDX case: ACPI tables are passed via TDVF and measured
- But what a safe configuration for all these 55+ tables?
  - Some tables, like DSDT, are complex, have many features, etc.
  - Most of them are not needed for CC guest

Solution: ACPI filter: minimal set of allowed ACPI tables

- XSDT, FACP, DSDT, FACS, APIC, SVKL, TDEL
- Command line option to extend the list
- **Code**:
  - https://github.com/intel/tdx/commit/4a120bb
  - https://github.com/intel/tdx/commit/6712f7b
  - https://github.com/intel/tdx/commit/a4239c1d2

Potential future work:

- Minimal ACPI configuration for CC guest
- Hardening of AML interpreter

Summary

- Hardening CC guest kernel applies to all CC solutions
- Core hardening mechanism are important to get right
- Only can be done as collaboration/partnership
  - Intel is ready to contribute, but we cannot do this alone

- What collaboration method is most effective?
  - Sending patches/doing discussions on coco mailing list vs lkml?
  - Once a month sync call with all the interested players to plan the work further?

References

- TDX guest hardening public documentation:
  - https://intel.github.io/ccc-linux-guest-hardening-docs/index.html
- Public Tools (audit & fuzzing):
  - https://github.com/intel/ccc-linux-guest-hardening
- Patches:
  - https://github.com/intel/tdx/tree/guest & other guest branches
- More detailed talk on CC guest kernel hardening aspects:
  - https://static.sched.com/hosted_files/lssna2022/74/LSS-HardeningLinuxGuestForCCC_2022.pdf

- Contact: elena.reshetova@intel.com

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

- No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document subject to open source software code (OSS) included in this document is licensed under the applicable OSS license from each respective licensor at: github.com/intel/tdx/tree/guest, https://github.com/intel/ccc-linux-guest-hardening, https://github.com/intel/ccc-linux-guest-hardening-docs

Backup

MSRs

- Trusted: TDX module controlled
  - Disallowed: #GP(0) or conditional #GP(0)
    - Examples: IS32_SMBASE, IA32_VMX_*, ..
  - Native: Context switch by HW
    - Examples: IA32_SPEC_CTRL, IA32_SYSENTER_*, IA32_FSBASE/GSBASE, ..
- Untrusted: TDX module inserts #VE or direct access via TDVMCALL
  - Audit & Fuzzing: Simple handling in code (bitmasking), pay attention to enabled features
  - Disable complex and unneeded MSRs: IA32_MTRR_*, IA32_MKTME_PARTITIONING, ...

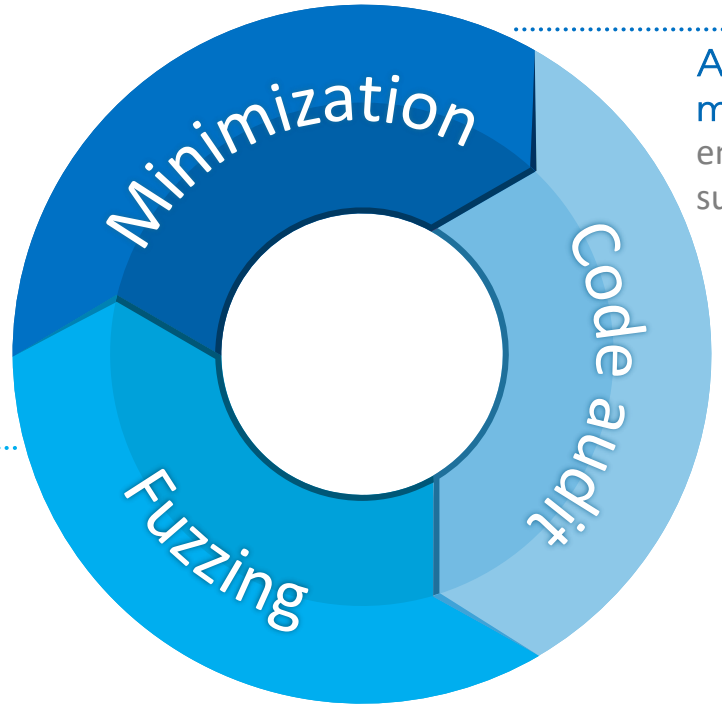
CPUIDs

- Trusted: TDX module controlled
  - Configured per attested params: XFAM, ATTRIBUTES, TD_PARAMS, CPUID_CONFIG
  - Native and Fixed
- Untrusted: TDX module inserts #VE or direct access via TDVMCALL
  - Disable unneeded: X86_FEATURE_CQM_LLC , X86_FEATURE_MBA, X86_FEATURE_TME
  - #VE handler only issues TDVMCALL for range 0x40000000 - 0x400000FF

KVM hypercalls & CPUIDs

- All untrusted:
  - TDX module inserts #VE or direct access via TDVMCALL
- Disable all hypercalls apart from KVM_HC_SEND_IPI
- Disable all CPUIDs apart from
  - KVM_FEATURE_NOP_IO_DELAY
  - KVM_FEATURE_PV_SEND_IPI
  - KVM_HINTS_REALTIME
- Code: 1 patch
  https://github.com/intel/tdx/commit/a768c6bea39012220ee5e0d8223636f59adbb5ff

Hardening strategy overview



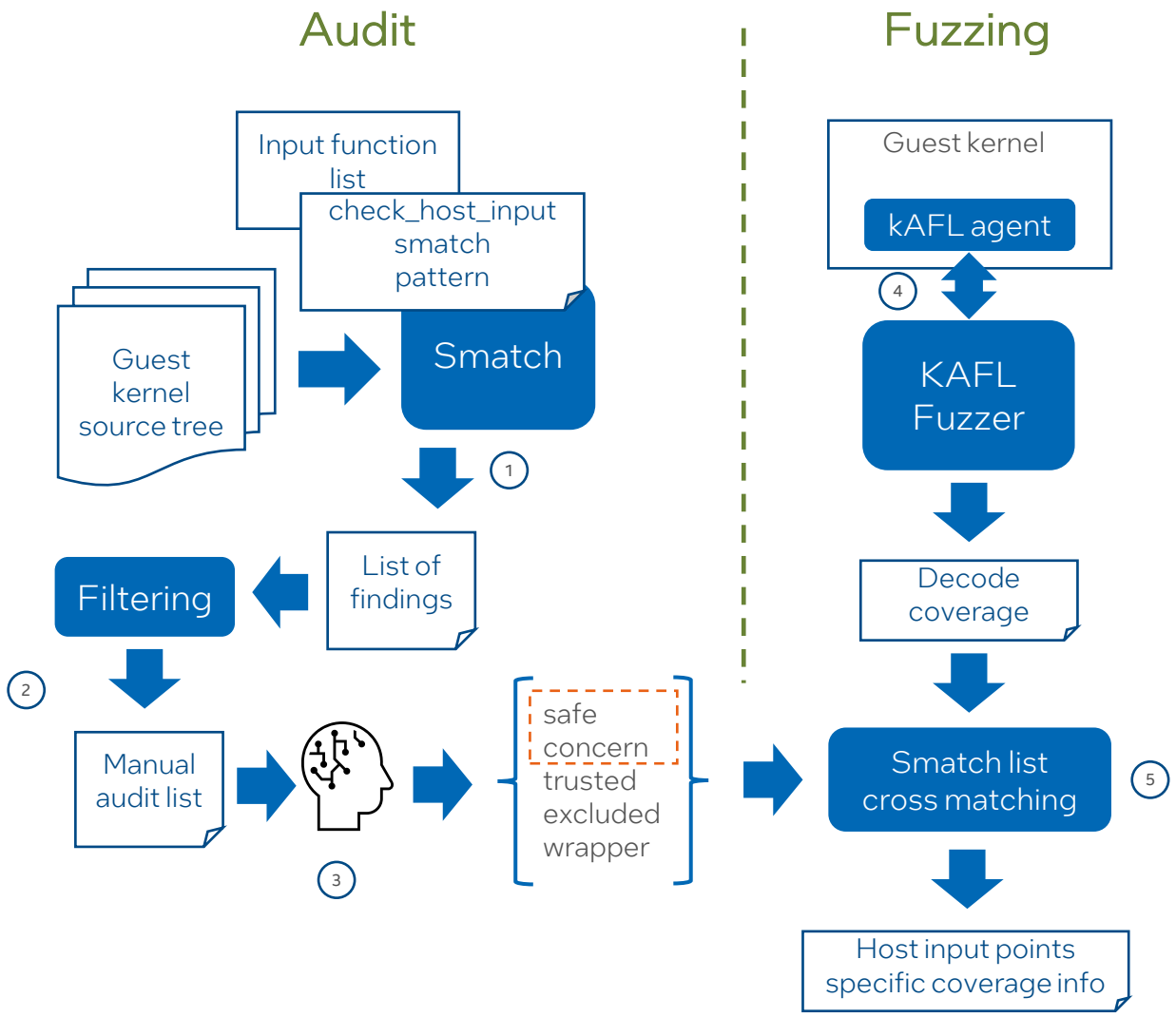
Attack surface minimization: enabled drivers, kernel subsystems, open I/O ports…

Static analyzer driven manual code audit to identify problematic places that might need to be disabled, target fuzzed or a code fix might be required

Wide scale and targeted fuzzing to exercise the exposed attack surface either in-breath or in-depth for more complex code locations

Iterative Approach

Hardening process for TDX guest code



Audit

1. Perform a smatch run on the target guest kernel source tree
2. Filter out code disabled by the driver filter
3. Perform manual audit of the results

Fuzzing

4. Run kAFL Fuzzer with several boot harnesses, collect code coverage and map to source (Qemu snapshot fuzzing + Intel PT trace dumps)
5. Cross reference generic code coverage against a list generated by smatch

Results of hardening: 5.15-rc1

Audit tag distribution

Audit results	# code locations
excluded	2505
wrapper	198
trusted	76
safe	1126
concern	81

Fuzzing coverage per audit tag type

Audit tag type	% of code locations	% of functions
safe	86%	87%
concern	92%	100%

Stable reproducible fuzzing findings

Fuzzing issue type	# of bugs
NULL pointer dereference	4
KASAN: user-memory-access	2
KASAN: out-of-bounds	7
kmalloc redzone oob	1
Supervisor write pagefault	2
Supervisor read pagefault	1
#GP	10
KASAN use-after-free	1

Existing functional patch fixes per area

area	# patches
MSIx	1
virtio-net	1
virtio-console	3
virtio-9p	1
virtio-pci-modern	1