Case Studies of Fuzzing with Xen

Tamas K Lengyel @tklengyel

Balint Varga-Perke
@buherator

Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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.

Your costs and results may vary.

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

TL;DR

KF/x: https://github.com/intel/kernel-fuzzer-for-xen-project

Full-VM snapshot fuzzer

Open-source (MIT)

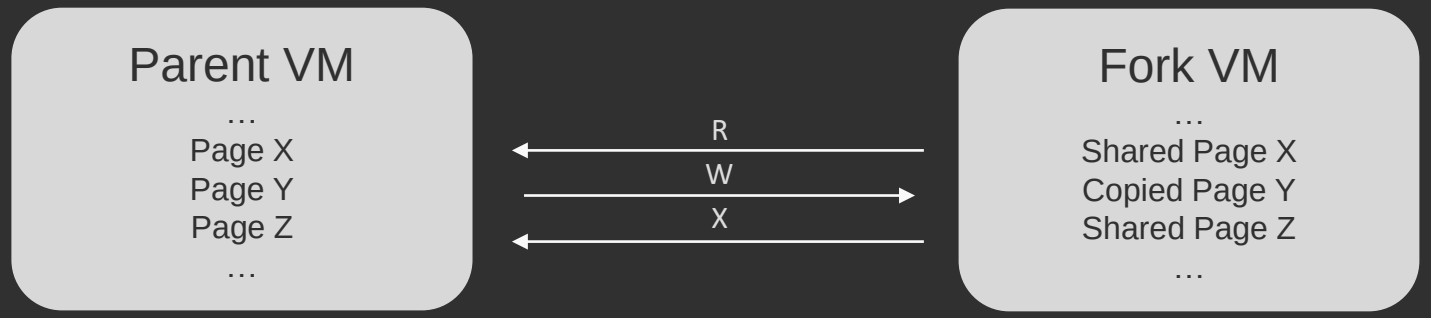
Integrates with AFL/AFL++

In this talk – the tale of two vastly different targets

- Linux Virtio
- Symantec Endpoint Protection

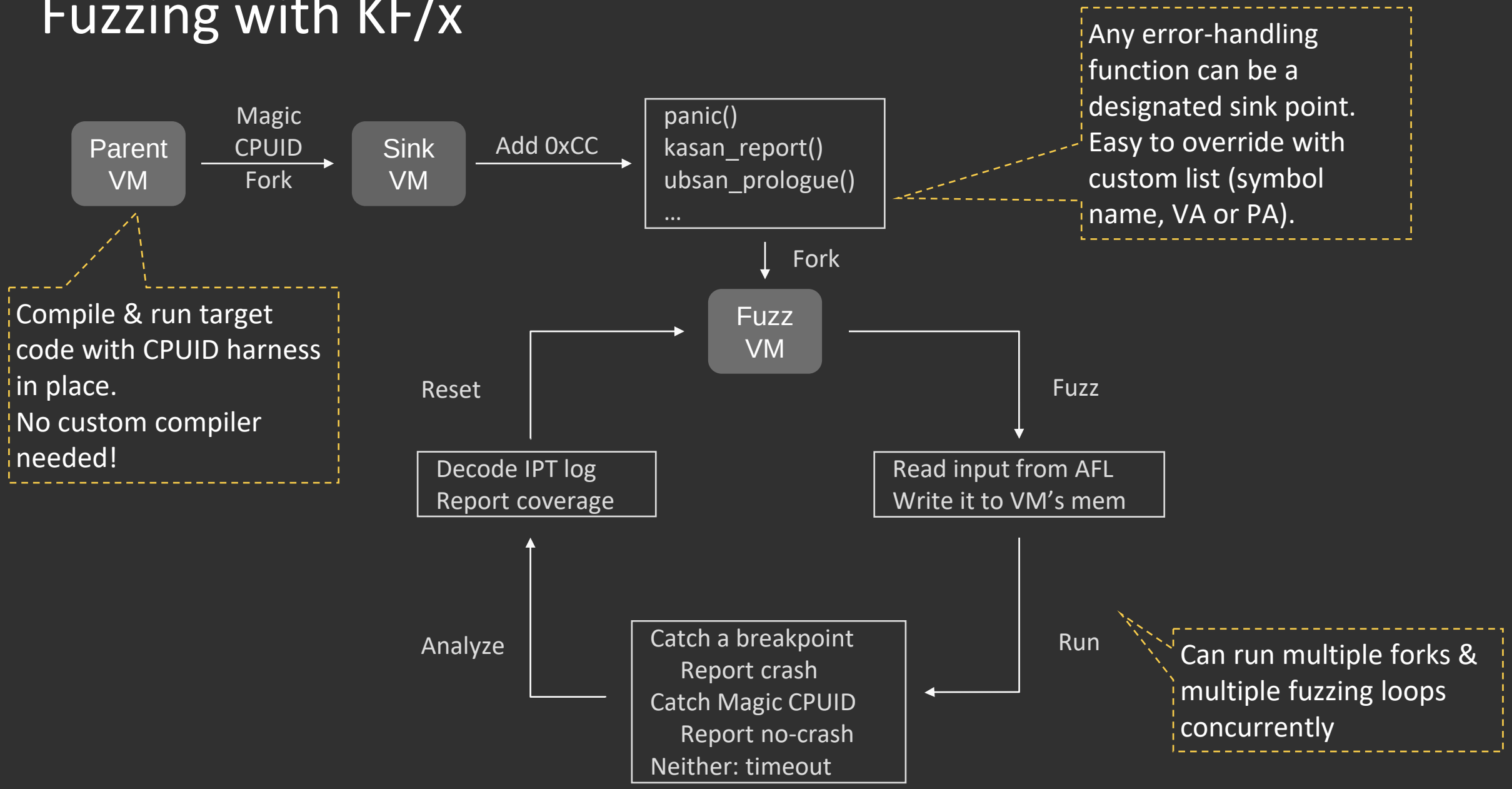
Xen VM forking

Copy-on-write memory in VM forks



Reset vCPU registers and free dirty pages

Fuzzing with KF/x



Design principles behind KF/x

Stable foundation

- All hypervisor components upstream (VM forking, introspection, IPT)

Reduced complexity

- No in-guest agent necessary

Flexible

- Components should be reusable to target any code running in the VM

Extendable

- Integrate with other fuzzers, use different harness mechanism, etc.

Limitations

No I/O in fork VMs

- No disk, no network, no screen, no console, no MMIO, no interrupts
- Single vCPU only

Bug enrichment features need to be compiled-in (ASAN, UBSAN)

Target needs to execute normally in a Xen VM

Fuzzing Virtio

Part of a larger kernel hardening project in preparation for TDX

See Linux Security Summit ‘21 talk by Elena Reshetova

Attacking guest from the host via shared memory (“DMA”)

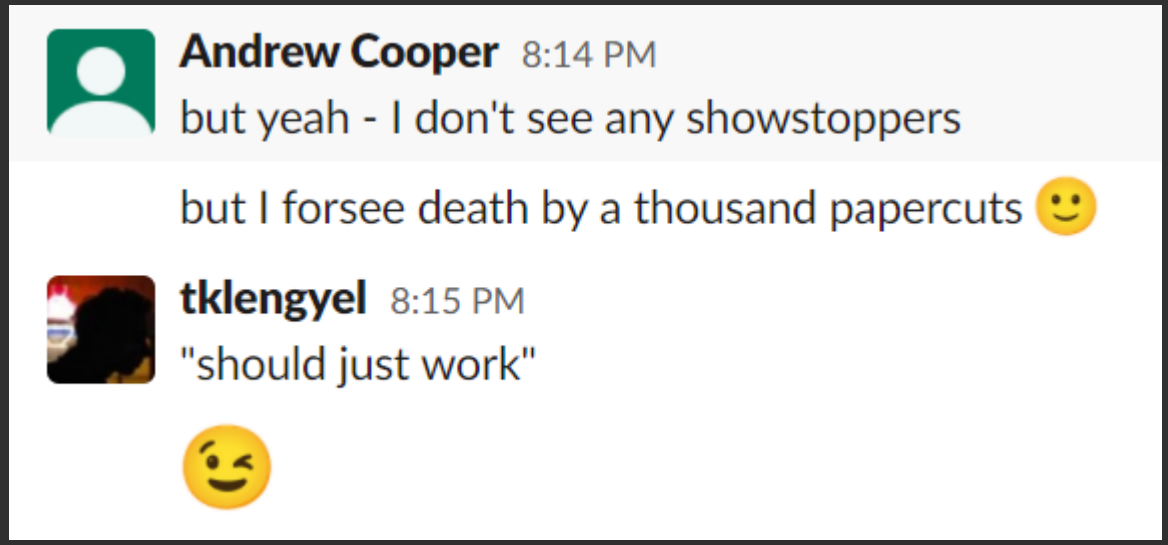
Xen doesn’t support Virtio – can’t use KF/x

- Unless .. !

VM transplantation!

The idea is simple:

1. Capture system state on KVM
2. Transfer it to Xen
3. Load
4. Fuzz!



Capturing the system state

During fuzzing we are running the VM forks with only:

- Memory
- CPU

We are in luck! QEMU QMP command dump-guest-memory

- Creates and ELF file detailing the memory map & memory contents
- Surprise undocumented feature: an ELF note has the CPU register state

We also need the magic CPUID pausing on KVM

```
diff --git a/arch/x86/kvm/cpuid.c b/arch/x86/kvm/cpuid.c
index 6bd2f8b830e4..f9855cf5eacd 100644
--- a/arch/x86/kvm/cpuid.c
+++ b/arch/x86/kvm/cpuid.c
@@ -1138,6 +1138,13 @@ int kvm_emulate_cpuid(struct kvm_vcpu *vcpu)

        eax = kvm_rax_read(vcpu);
        ecx = kvm_rcx_read(vcpu);
+
+       if ((vcpu->run->kfx.cpuid && eax == vcpu->run->kfx.cpuid) || (!vcpu->run->kfx.cpuid && eax == KVM_EXIT_KFX_CPUID)) {
+               vcpu->run->exit_reason = KVM_EXIT_KFX_CPUID;
+               kvm_skip_emulated_instruction(vcpu);
+               return 0;
+       }
```

KVM patch to exit on magic CPUID

QEMU patch to pause VM

```
--- a/target/i386/kvm/kvm.c
+++ b/target/i386/kvm/kvm.c
@@ -4219,6 +4219,8 @@ void kvm_arch_pre_run(CPUState *cpu, struct kvm_run *run)

         qemu_mutex_unlock_iothread();
     }
+
+    run->kfx.cpuid = kvm_kfx_cpuid;
 }

 MemTxAttrs kvm_arch_post_run(CPUState *cpu, struct kvm_run *run)
@@ -4594,6 +4596,12 @@ int kvm_arch_handle_exit(CPUState *cs, struct kvm_run *run)
         ioapic_eoi_broadcast(run->eoi.vector);
         ret = 0;
         break;
+    case KVM_EXIT_KFX_CPUID:
+        fprintf(stderr, "KVM: KF/x CPUID caught!\n");
+        kvm_cpu_synchronize_state(cs);
+        vm_stop(RUN_STATE_PAUSED);
+        ret = 0;
+        break;
```

Loading state on Xen

```
root@t2:/shared/cfg# cat transplant.cfg
arch = 'x86_64'
name = "transplant"
memory = 4000
vcpus = 1
hap = 1
nomigrate = 1
type = "hvm"
vga = "none"
vnc = 0
vmtrace_buf_kb = 65536
```

1. Create “empty” VM
2. Load memory according to memory map
3. Load CPU register state
   - Fix mismatch between segment attribute byte format in ELF vs Xen

https://en.wikipedia.org/wiki/Segment_descriptor

The x86 and x86-64 segment descriptor has the following form:[3]

31	—	24	23	22	21	20	19	—	16	15	14	13	12	11	10	9	8	7	—	0
Base Address[31:24]			G	D/B	L	AVL	Segment Limit[19:16]			P	DPL		1	Type		C/E	R/W	A	Base Address[23:16]	
Base Address[15:0]										Segment Limit[15:0]										

```
/*
 * Xen expects segment attribute bits in the following format
 */
union arb
{
    struct
    {
        uint16_t type:4; // 0-3
        uint16_t s:   1; // 4
        uint16_t dpl: 2; // 5-6
        uint16_t p:   1; // 7
        uint16_t avl: 1; // 8
        uint16_t l:   1; // 9
        uint16_t db:  1; // 10
        uint16_t g:   1; // 11
        uint16_t pad: 4; // 12-15
    };
    uint16_t _u;
} arb;
```

It’s alive!

```
root@t2:/shared/5.10/capture/snapshot-0xf9f9bcaf0c93407c# xl create -p -e /shared/cfg/transplant.cfg
Parsing config from /shared/cfg/transplant.cfg
root@t2:/shared/5.10/capture/snapshot-0xf9f9bcaf0c93407c# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0 12096     8     r-----  177430.3
transplant                                  37  3999     1     --p---       0.0
root@t2:/shared/5.10/capture/snapshot-0xf9f9bcaf0c93407c# xen-transplant 37 ./regs.csv ./memmap ./vmcore
Set vCPU context: success
Loading memory from file offset: 0x0 to memory offset: 0xfffc0000 Size: 0x40000
Loaded pages: 0 Failed: 64
Loading memory from file offset: 0x40000 to memory offset: 0x0 Size: 0xa0000
Loaded pages: 160 Failed: 0
Loading memory from file offset: 0xe0000 to memory offset: 0xc0000 Size: 0x3c940000
Loaded pages: 248128 Failed: 0
Loading memory from file offset: 0x3ca20000 to memory offset: 0xfb000000 Size: 0x1000000
Loaded pages: 0 Failed: 4096
VM transplanting successful
root@t2:/shared/5.10/capture/snapshot-0xf9f9bcaf0c93407c# stepper --domid 37 --limit 10
Init vmi, init_events: 1 init_paging 1 domain (null) domid 37 json (null) kvmi (null)
    0: ffffffff819f69f2  movzx ebx, word ptr [rbx + 0xe]
    1: ffffffff819f69f6  mov rdi, qword ptr [rbp - 0x70]
    2: ffffffff819f69fa  call 0xc474758
    3: ffffffff813e7150  push rbp
    4: ffffffff813e7151  mov rbp, rsp
    5: ffffffff813e7154  mov r8, qword ptr [rbp + 8]
    6: ffffffff813e7158  cmp rdi, -5
    7: ffffffff813e715c  ja 0xca84052
    8: ffffffff813e715e  movabs rax, 0xffff7fffffffffff
    9: ffffffff813e7168  cmp rdi, rax
   10: ffffffff813e716b  jbe 0xca84043
```

Back to Virtio

We have a way to save, transplant & fuzz

Just have to figure out what to fuzz..

Virtio will be used for all I/O on TDX 1.0

- Disk, network, console

That’s a lot of different code-paths to cover

- Does anyone even know all the different ways Virtio code is reached?

Retargeting existing code

Had a similar challenge while fuzzing xHCI

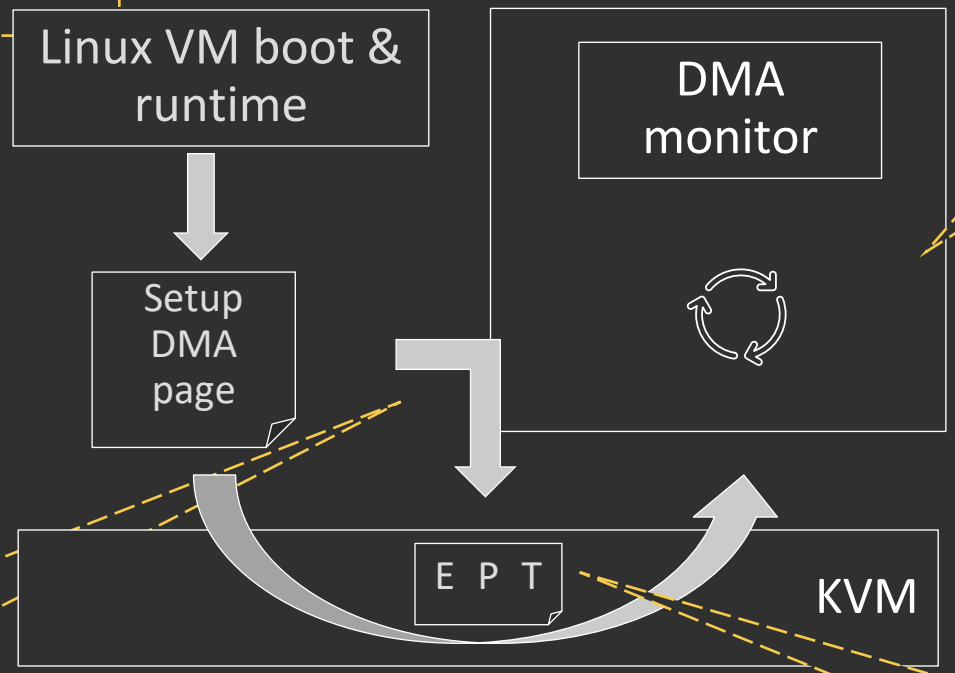
Made a tool (dmamonitor) that can hook Linux's DMA API

- Hook dma_alloc_attrs with VMI
- Remove EPT permission from DMA pages
- Log RIP when EPT fault triggers with read-violation

Could really use it here too..

- We are in luck: https://github.com/KVM-VMI/kvm-vmi

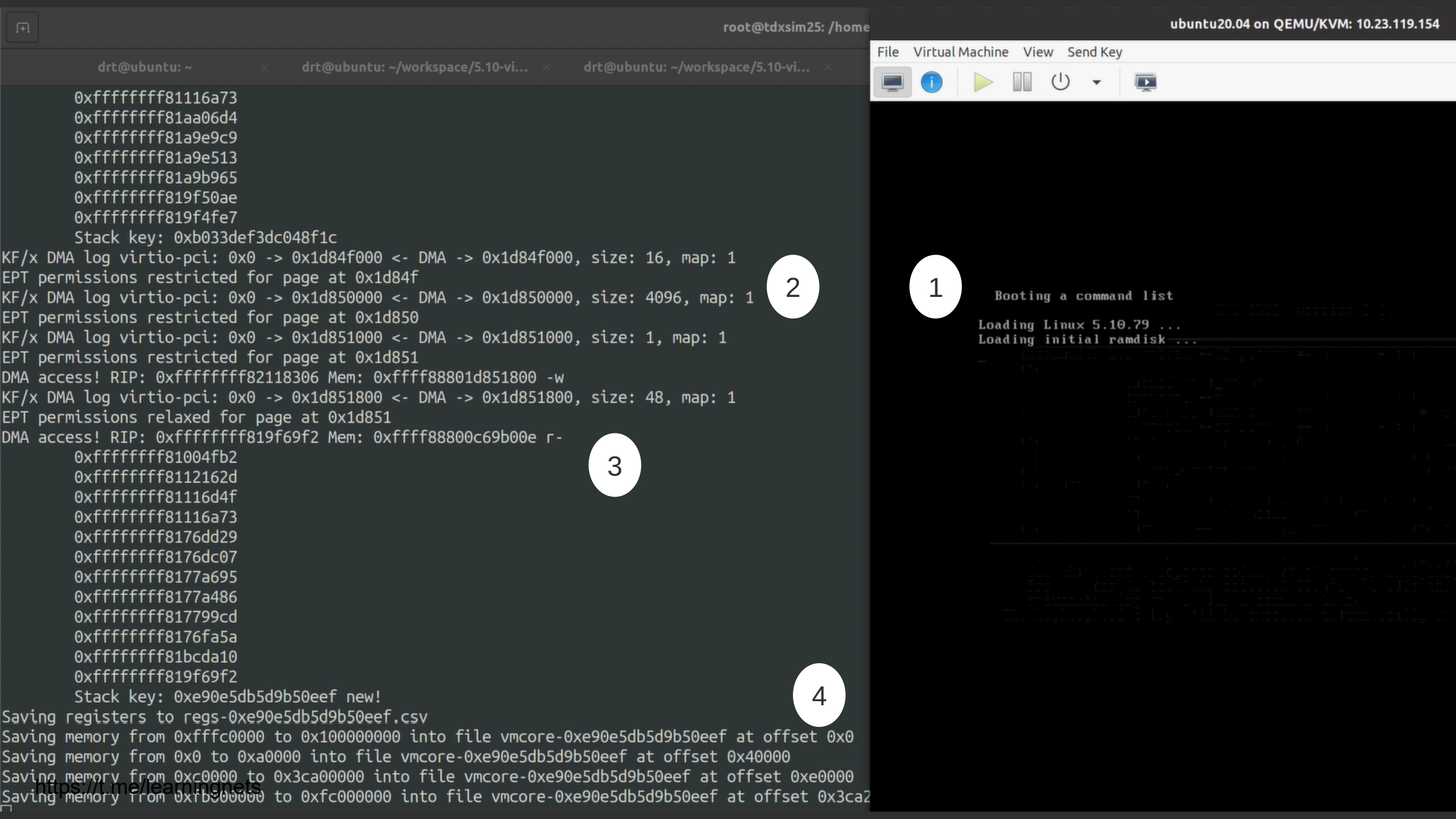
1. Boot Linux with DMA debug patched to have dummy API for easier hooking & force Virtio to go through DMA API



4. Handle EPT read-fault
Unwind stack & hash return pointers
If hash is new, snapshot

2. DMA hook triggers for DMA page use
Change EPT permission

3. EPT fault triggers when Virtio is fetching data from page


root@tdxsim25: /home
drt@ubuntu: ~
drt@ubuntu: ~/workspace/5.10-vi...
drt@ubuntu: ~/workspace/5.10-vi...
0xffffffff81116a73
0xffffffff81aa06d4
0xffffffff81a9e9c9
0xffffffff81a9e513
0xffffffff81a9b965
0xffffffff819f50ae
0xffffffff819f4fe7
Stack key: 0xb033def3dc048f1c
KF/x DMA log virtio-pci: 0x0 -> 0x1d84f000 <- DMA -> 0x1d84f000, size: 16, map: 1
EPT permissions restricted for page at 0x1d84f
KF/x DMA log virtio-pci: 0x0 -> 0x1d850000 <- DMA -> 0x1d850000, size: 4096, map: 1
EPT permissions restricted for page at 0x1d850
KF/x DMA log virtio-pci: 0x0 -> 0x1d851000 <- DMA -> 0x1d851000, size: 1, map: 1
EPT permissions restricted for page at 0x1d851
DMA access! RIP: 0xffffffff82118306 Mem: 0xffff88801d851800 -w
KF/x DMA log virtio-pci: 0x0 -> 0x1d851800 <- DMA -> 0x1d851800, size: 48, map: 1
EPT permissions relaxed for page at 0x1d851
DMA access! RIP: 0xffffffff819f69f2 Mem: 0xffff88800c69b00e r-
0xffffffff81004fb2
0xffffffff8112162d
0xffffffff81116d4f
0xffffffff81116a73
0xffffffff8176dd29
0xffffffff8176dc07
0xffffffff8177a695
0xffffffff8177a486
0xffffffff817799cd
0xffffffff8176fa5a
0xffffffff81bcda10
0xffffffff819f69f2
Stack key: 0xe90e5db5d9b50eef new!
Saving registers to regs-0xe90e5db5d9b50eef.csv
Saving memory from 0xfffc0000 to 0x100000000 into file vmcore-0xe90e5db5d9b50eef at offset 0x0
Saving memory from 0x0 to 0xa0000 into file vmcore-0xe90e5db5d9b50eef at offset 0x40000
Saving memory from 0xc0000 to 0x3ca00000 into file vmcore-0xe90e5db5d9b50eef at offset 0xe0000
Saving memory from 0xfb000000 to 0xfc000000 into file vmcore-0xe90e5db5d9b50eef at offset 0x3ca2
2
ubuntu20.04 on QEMU/KVM: 10.23.119.154
File
Virtual Machine
View
Send Key
1
Booting a command list
Loading Linux 5.10.79 ...
Loading initial ramdisk ...
3
4
https://t.me/learningnets

No end-harness

We don’t need any!

1. Transplant snapshot
2. Fork
3. Singlestep up to 300k
4. Check which of the stack return pointers was reached
5. Inject breakpoint to transplant
6. Fuzz!

```
DMA access! RIP: 0xffffffff819f69f2 Mem: 0xffff88800c69b00e r-
        0xffffffff81004fb2
        0xffffffff8112162d
        0xffffffff81116d4f
        0xffffffff81116a73
        0xffffffff8176dd29
        0xffffffff8176dc07
        0xffffffff8177a695
        0xffffffff8177a486
        0xffffffff817799cd
        0xffffffff8176fa5a
        0xffffffff81bcda10
        0xffffffff819f69f2
        Stack key: 0xe90e5db5d9b50eef new!
```

Results with hardened 5.15-rc6

126,061 DMA accesses observed during boot and basic functioning

13 unique DMA access sites

738 unique call-chains lead to DMA access

70 snapshots fuzzed based on top-5 stack frame uniqueness

7,567,463,809 fuzzing cycles completed (in 2 weeks)

0 issues found (no KASAN/UBSAN/panic/oops)

13 snapshots were found to have hangs when fuzzed

54 snapshots had less then 5 paths discovered

No bugs?

We weren't the first

- Check out VIA: Analyzing Device Interfaces of Protected Virtual Machines
- They fuzzed 5.10 and already reported the sanitizer bugs & got them fixed
- We were able to catch some of the same bugs they found when we targeted 5.10

I still consider this a win

- Tools & techniques all open-sourced & anyone is welcome to replicate
- KF/x target setup & fuzzing can now happen at different systems
- No longer need to setup your target on Xen (can use QEMU/KVM/Simics)

Thanks to the whole lot of folks!

Andrew Cooper, Sundaram Arumugasundaram, Mostafa Elsaid, Andi Kleen, Neelima Krishnan, Sathyanarayanan Kuppuswamy, Lukasz Odzioba, Sebastian Osterlund, Elena Reshetova, Carlos Villavicencio Sanchez, Casey Schaufler, Steffen Schulz, Mathieu Tarral

Antivirus fuzzing

- Widespread technology
- Complex parsers implemented in C/C++
- Remotely reachable attack surface
- High privileges
- Prior work:

Antivirus (in)Security, Attacking Antivirus, P0 massacre, Sophail, Nightmare, The AV Hacker's Handbook, REing Defender, tons of privescs, blackhats, and many others (sry if I missed yours!)

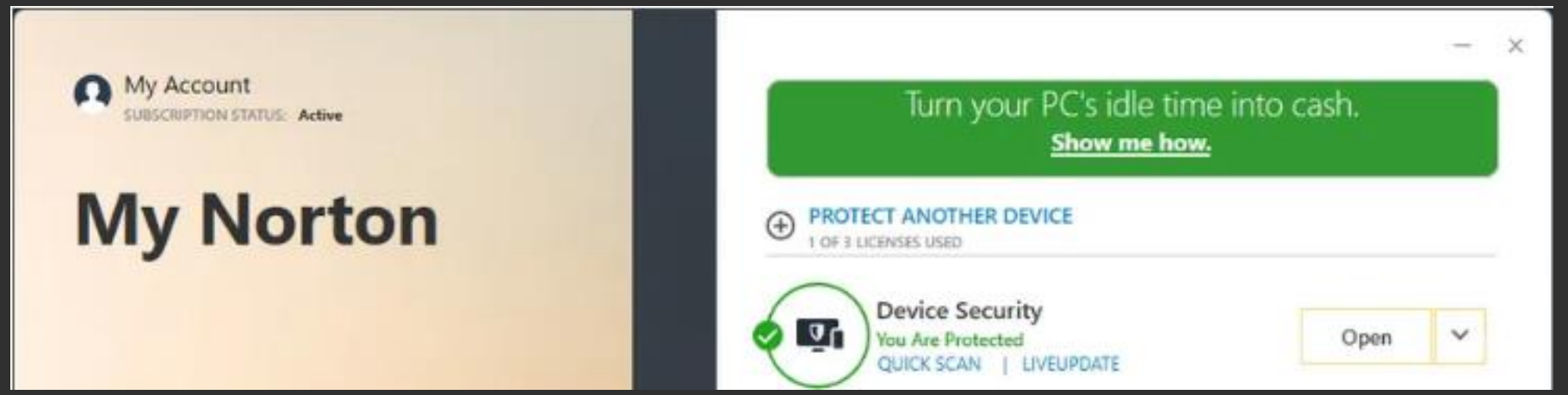
Antivirus fuzzing

- Complex software
- Binary-only (mostly)
- Performance
- Inspectability
- State
- Diversity
  - Products
  - Formats

KF/x vs. AV

- Reusable harness
- Full inspectability
- Full-system fuzzing
- **Written documentation** + well-known platform

Symantec Endpoint Protection



Symantec Endpoint Protection

- Tamper protection
  - Can be disabled, some memory still not accessible
- COM-like architecture
  - Complex inter-module dependencies (writeup soon)
  - COM is obfuscation: No export symbols, no typelib, indirect calls …
- OS interference

Proof-of-Concept

- Multiple bugs discovered by Tavis Ormandy in 2016.
  - "Decomposers"
  - Unrar, dec2lha, libmspack, … -> ccScanW.dll
- Try to rediscover #823 – "PowerPoint misaligned stream-cache remote stack buffer overflow"
  - Easy to modify nasm PoC
  - /GS still not applied to all (non-trivial) functions #YOLO

Proof-of-Concept

- #823 allowed quick identification of the relevant parser function and I/O functions
- KF/x tuning
  - Interrupt masking
  - sinks.h -> “KiDispatchException” (resolved by Volatility)
  - if (addr & 0x80000000) return;
- Large test case
  - SEP reads data in 8k chunks
  - 6 reads necessary before the bug triggers (no disk in KF/x!)

Rediscovering CVE-2016-2209 with KF/x

Handling large inputs

- Large inputs are not ideal for fuzzing in general
  - Performance
  - Process state
- Mocking I/O
  - Tried it, wouldn't recommend…
- RAM disk works (on Windows too)
- Chained fuzzing stages
- Other traps: swapping, console output, network, …

Other parsers?

- Rabbit Hole Ghidra extension
  - Per-function cumulative cyclomatic complexity
- Ghidra Cpp Class Analyzer
  - RTTI info
- Magic numbers
- New target: 7zip decomposer
  - Magic number + high complexity + reachable from C7zEngine class members

```
*************************************************************
* const C7zEngine::vftable
*************************************************************
C7zEngine::vftable                                        XRE

    C7zEngin...


10     addr     C7zEngine::operator_delete_cc3

10     addr     C7zEngine::7zEngine_1007ba10_cc1
10     addr     C7zEngine::7zEngine_1006ecb0_cc3
10     addr     C7zEngine::7zEngine_1007b9c0_cc4
10     addr     C7zEngine::7zEngine_1006ec90_cc2
10     addr     C7zEngine::7zEngine_1007bff0_cc2390
10     addr     C7zEngine::7zEngine_1006e410_cc1
10     addr     C7zEngine::7zEngine_1007b7c0_cc3
```

```
▼ ƒ Incoming References - 7z_magic_matcher_cc2267
  ▼ ƒ 7z_magic_wrapper_bde90_cc2284
    ▼ ƒ FUN_100985b0_cc2302
      ▼ ƒ 7z_hash_initiator_cc2310
          ƒ 7zEngine_1007bff0_cc2390
```

“While this fuzzer runs, I will…”[1]

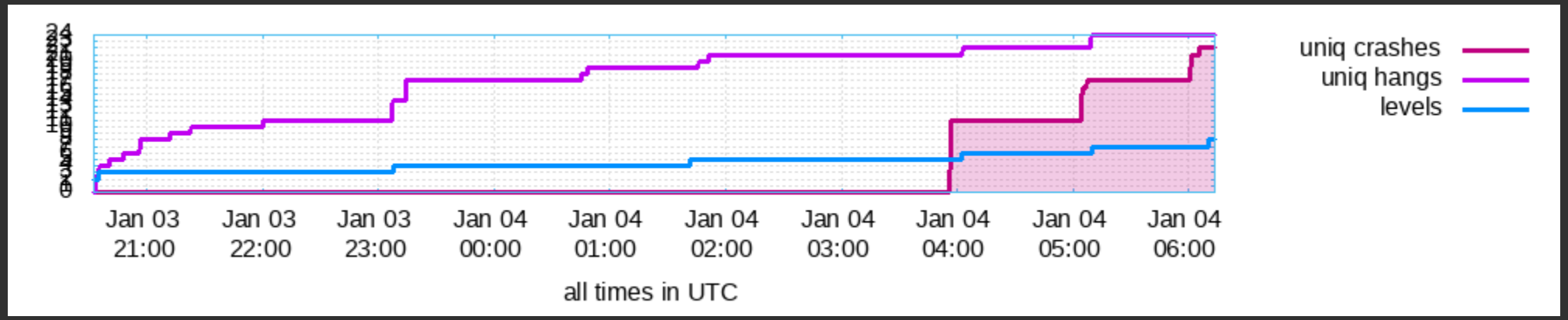
- Corpus: 1 file, some txt 7z’d with default options
- AFL++, no knowledge about the file format
- Coverage tracking with Intel PT
- Single core
- < 24h runtime
  - ~9M execs ~400-500 exec/s (not a perf talk sry)
  - No hang elimination (“loose paths”, timeout optimization, …)

[1] How to FAIL at Fuzzing, Prospector

Fuzzing Symantec Endpoint Protection's 7z parser with KF/x

7z bug

- Controlled heap overflow
- Detection can be improved by enabling page heap
- Silently fixed?
  - Symantec->Broadcom didn’t make investigation easy



Mandatory AFL graph

Using REVEN to triage the 7z parsing heap overflow discovered with KF/x

Modularity

- KF/x follows the Unix philosophy
- AFL’s SHMAP became a de-facto standard
- Forkserver-based intergration with LibAFL was trivial
  - Timeout/signal handling needed improvement (I/O fail -> hang)
  - Independent development of generators, mutators, etc.
  - In-memory input passing needs little more work

LibAFL

```
[root@zero1 /home/b/LibAFL/fuzzers/forkserver_simple ]% ./target/release/forkserver_simple "/home/b/kfx2112/kfx" /home/
b/7z/input --  --harness breakpoint --domain sym_setup --input @@ --input-limit 8191 --ptcov --start-byte 0xA1 --json /
home/b/win8/dummy8.json --address 0x38A3200 -F /tmp/kfx.log --debug
All right - fork server is up.
Forkserver Options are not available.
Loading file "/home/b/7z/input/test1.7z" ...
[Stats #0] run time: 0h-1m-0s, clients: 1, corpus: 0, objectives: 0, executions: 0, exec/sec: 0
[Testcase #0] run time: 0h-1m-0s, clients: 1, corpus: 1, objectives: 0, executions: 1, exec/sec: 0
[LOG Debug]: Loaded 1 initial testcases.
We imported 1 inputs from disk.
[Stats #0] run time: 0h-1m-0s, clients: 1, corpus: 1, objectives: 0, executions: 1, exec/sec: 0
[Testcase #0] run time: 0h-1m-0s, clients: 1, corpus: 2, objectives: 0, executions: 2, exec/sec: 0
[Stats #0] run time: 0h-1m-0s, clients: 1, corpus: 2, objectives: 0, executions: 2, exec/sec: 0
[Testcase #0] run time: 0h-1m-0s, clients: 1, corpus: 3, objectives: 0, executions: 3, exec/sec: 0
```

Other use-cases

- AV
  - Kernel components
  - Memory scanners
  - DPI/DLP/IPS features
- Games / Anti-cheat
- Sandbox escapes
  - Nyx Fuzzer, Fuzzy Snapshots of Firefox IPC

Summary

- Trivial vulnerabilities could remain hidden due to the lack of proper tools
- VM Introspection is a game changer in vulnerability discovery/analysis
- KF/x is an easy to integrate VMI-based fuzzing harness

Thanks!

KF/x: https://github.com/intel/kernel-fuzzer-for-xen-project

Tamas K Lengyel @tklengyel

Balint Varga-Perke @buherator