Distros and Syzkaller - Why bother?

1. What? Find out how distros and others are using Syzkaller and other fuzzers
2. Why? To collaborate to enhance distros development and release process (build a better kernel)
3. How? Continuous Integration (CI)
How to make Syzkaller part of distro release process?

Stable tree merge?  
New Feature?  
New Release?  
Bug fix?

Jenkins kernel rpm build w/ Syzkaller needed config settings  
KASAN, KCOV, "=y"

Dedicated Syzkaller CI Servers  
running distro's next release candidate

New crash?  
New Bug?

Auto bug create tool (cron job)

Today:  
1) kernel build  
2) dracut  
3) Halt Syzkaller;  
4) Re-Start Syzkaller w/ bzImage & initramfs

New Jenkins Build?  
1) Halt Syzkaller;  
2) install kernel rpm to VM;  
3) Re-Start Syzkaller w/ bzImage & initramfs from VM

New Bug?

Bug data base primed with existing bugs to avoid Dups

workdir/crashes/*
workdir/corpus/*
Example of Syzkaller benefit (rds_sendmsg bug regression)

• Syzkaller found this rds_sendmsg bug:
  – KASAN: stack-out-of-bounds Read in rds_sendmsg

• Bug fixed by this Upstream commit:
  – 14e138a Thu Dec 21 20:17:04 2017 -0800 RDS: Check cmsg_len before dereferencing CMSG_DATA

• Commit is also in our distro release.

• Weekly Syzkaller runs showed that the rds_sendmsg bug was back in our distro release. How could that be?

• Yes, commit 14e138a is in our distro release, but we had overlaid it with new code.

• Syzkaller found the regression!
What are others doing?

• How do others track Syzkaller repo?
  - we pull Stable tree merges to build our distro release
  - Syzkaller tracks latest Upstream
What are others doing? (continued)

• We run into these types of Syzkaller build errors as a result of our distro release missing latest kernel defines:
  − Syzkaller build of reproducer C program fails (e.g. IFLA_HSR_SLAVE1 missing from if_link.h). Problem can show up after hours of testing.
  − Syzkaller build fails (e.g. tools/syz-env/env.go:34:12: undefined: osutil.SystemMemorySize)

• To come up with a Syzkaller build that works with our distro release requires some intervention.
What are others doing? (continued)

• What is the strategy that others are using to upgrade Syzkaller? Monthly Syzkaller update? Quarterly?
Wish List

• Syzkaller repo tag corresponding to Stable tree tag.
• tarball of Syzbot reproducer C programs.
  ─ Perfect for regression smoke test.
  ─ Could cut test time!
What other types of fuzzing do distros and others use?

- we fuzz MSRs, Control Regs, Debug Regs, etc with nano-VM (minimal KVM ioctl).
- what about qemu fuzzing?
- what about PCI fuzzing?
- how to add code coverage for other types of fuzzing?
- what about Intel vs AMD and other Architectures?
- e.g. Syzkaller AMD-only bug:
  - kernel BUG at arch/x86/kvm/x86.c:LINE!
  - vmload ← svm_vcpu_run+0xa83
  - https://groups.google.com/forum/#!searchin/syzkaller/kvm$20amd$20cpu/syzkaller/bIntrLGt2JA/SbRvpM6oCAAJ
Conclusion – How can we collaborate?

• Continue discussions through Syzkaller google groups?
• emails?
• What else?
References

- https://github.com/google/syzkaller
- https://github.com/google/syzkaller/blob/master/docs/linux/setup.md
- kernel: add kcov code coverage: https://lwn.net/Articles/671640/
- kcov patch: https://lkml.org/lkml/2016/1/25/475
- https://github.com/google/syzkaller/blob/master/executor/test_linux.h
Backup Slides
MSR fuzzing pseudo-code

Goal: allow any MSR to be written with any bit pattern to ensure that the host does not crash

WRMSR — Write to Model Specific Register
Writes the contents of registers EDX:EAX into the 64-bit model specific register (MSR) specified in the ECX register.

```
./msrtest -m 0xc0000080 -w -a 0x1 -d 0x0
```

// msrtest calls test_one() to run nano-VM to write MSR 0xc0000080 (ecx) with eax=1 and edx=0
// KVM_SET_REGS ioctl will set RCX, RAX, & RDX

```
00000000 <.data>: // nano-VM: 3 byte "kernel" to be launched by KVM_RUN ioctl in test_one()
0: 0f 30           wrmsr
2: f4              hlt
```

// test_one() called by msrtest
// codep: pointer to above 3 byte "kernel"
// codesz: 3
if ((res = test_one(32, codep, codesz-1, 0, KVM_EXIT_HLT, false))) // from syzkaller/executor/test_linux.h
    return res;
...

exit_reason: 5
MSR fuzzing pseudo-code (continued)

static int test_one(int text_type, const char* text, int text_size, int flags, unsigned reason, bool check_rax)       // from syzkaller/executor/test_linux.h
{
    int vmfd = ioctl(kvmfd, KVM_CREATE_VM, 0);
    int cpufd = ioctl(vmfd, KVM_CREATE_VCPU, 0);
    int cpu_mem_size = ioctl(kvmfd, KVM_GET_VCPU_MMAP_SIZE, 0);

    // do mmap()

    struct kvm_text kvm_text;
    kvm_text.typ = text_type;
    kvm_text.text = text;
    kvm_text.size = text_size;

    // ioctls used: KVM_SET_USER_MEMORY_REGION, KVM_GET_SREGS, KVM_SET_CPUID2, KVM_SET_SREGS
    // KVM_SET_REGS ioctl will set regs RCX, RAX & RDX for wrmsr
    if (syz_kvm_setup_cpu(vmfd, cpufd, (uintptr_t)vm_mem, (uintptr_t)&kvm_text, 1, flags, 0, 0))
        error ...
    ioctl(cpufd, KVM_RUN, 0);

    ioctl(cpufd, KVM_GET_REGS, &regs);

    if (cpu_mem->exit_reason != reason)
        error ...
}

What is Syzkaller? (based on Dmitry’s slide)
Hardware and Software
Engineered to Work Together