syzkaller

the next gen kernel fuzzer

Qualcomm Mobile Security Summit 2017

Dmitry Vyukov (dvyukov@), Google

Agenda

- Kernel sanitizers (KASAN, KMSAN, KTSAN)
- Why new fuzzer? How is it better?
- Operational side
- Tutorial
- Extending syzkaller to fuzz new drivers

KASAN (KernelAddressSanitizer)

Fast and comprehensive solution for both **UAF** and **OOB**.

- based on compiler instrumentation
- detects both OOB writes and OOB reads
- detects OOB on heap, stack and globals
- strong UAF detection
- double-free detection
- detects bugs at the point of occurrence
- prints informative reports

Upstream: CONFIG_KASAN + gcc 5.0+

KASAN Report (CVE-2013-7446)

```
BUG: KASan: use-after-free in remove_wait_queue
Write of size 8 by task syzkaller_execu/10568
Call Trace:
 list_del include/linux/list.h:107
 __remove_wait_queue include/linux/wait.h:145
 remove_wait_queue+0xfb/0x120 kernel/sched/wait.c:50
 ...
 SYSC_exit_group kernel/exit.c:885
Allocated:
 kmem_cache_alloc+0x10d/0x140 mm/slub.c:2517
 sk_prot_alloc+0x69/0x340 net/core/sock.c:1329
 sk_alloc+0x33/0x280 net/core/sock.c:1404
 ...
 SYSC_socketpair net/socket.c:1281
Freed:
 kmem_cache_free+0x161/0x180 mm/slub.c:2745
 sk_prot_free net/core/sock.c:1374
 sk_destruct+0x2e9/0x400 net/core/sock.c:1452
 ...
 SYSC_write fs/read_write.c:585
```

KMSAN (KernelMemorySanitizer)

Detects uses of **uninitialized memory.**
In the context of security: **information leaks** (local or remote).

Most likely it does not what you think it does:
not loads of not-stored-to variables, but uses of uninitialized values (see backup)
No false positives and almost no false negatives.

Not upstreamed yet (on github): CONFIG_KMSAN + clang

KMSAN Report

```
BUG: KMSAN: use of unitialized memory in nf_ct_frag6_gather
Call Trace:
 nf_ct_frag6_gather+0xf5a/0x44a0 net/ipv6/nf_conntrack_reasm.c:577
 ipv6_defrag+0x1d9/0x280 net/ipv6/netfilter/nf_defrag_ipv6_hooks.c:68
 nf_hook_entry_hookfn ./include/linux/netfilter.h:102
 nf_hook_slow+0x13f/0x3c0 net/netfilter/core.c:310
 ...
 SyS_sendto+0xbc/0xe0 net/socket.c:1664

Origin:
 __alloc_skb+0x2cd/0x740 net/core/skbuff.c:231
 alloc_skb ./include/linux/skbuff.h:933
 alloc_skb_with_frags+0x209/0xbc0 net/core/skbuff.c:4678
 sock_alloc_send_pskb+0x9ff/0xe00 net/core/sock.c:1903
 ...
 SyS_sendto+0xbc/0xe0 net/socket.c:1664
```

KTSAN (KernelThreadSanitizer)

Detects **data races** (two unsynchronized accesses in different threads, at least one is a write).

Kernel data races represent **security threat**:

- TOCTOU
- uninit/wrong credentials
- racy use-after-frees
- double frees caused by races
- the most frequent type of bug

Not upstreamed yet (prototype on github): CONFIG_KTSAN

KTSAN Report (CVE-2015-7613)

```
ThreadSanitizer: data-race in ipc_obtain_object_check

Read at 0x123 of size 8 by thread 234 on CPU 5:
 ipc_obtain_object_check+0x7d/0xd0 ipc/util.c:621
 msq_obtain_object_check ipc/msg.c:90
 msgctl_nolock.constprop.9+0x208/0x430 ipc/msg.c:480
 SYSC_msgctl ipc/msg.c:538

Previous write at 0x123 of size 8 by thread 567 on CPU 4:
 ipc_addid+0x217/0x260 ipc/util.c:257
 newque+0xac/0x240 ipc/msg.c:141
 ipcget_public ipc/util.c:355
 ipcget+0x202/0x280 ipc/util.c:646
 SYSC_msgget ipc/msg.c:255

Also: locked mutexes, thread creation stacks, allocation stack, etc.
```

fuzzing + sanitizers = perfect fit!

syzkaller

Existing Fuzzers

trinity/iknowthis in essence:

```
for (;;) { syscall(rand(), rand(), rand()); }
```

Do know argument types, so more like:

```
for (;;) { syscall(rand(), rand_fd(), rand_addr()); }
```

Downsides:

- tend to find shallow bugs
- frequently no reproducers
- poorly suitable for regression testing

Coverage-guided Fuzzing

Code coverage guiding:

- corpus of "interesting" inputs
- mutate and execute inputs from corpus
- if inputs gives new code coverage, add it to corpus

Advantages:

- turns exponential problem into linear (more or less)
- inputs are reproducers
- corpus is perfect for regression testing

But how to apply it to kernel?

Kernel Code Coverage

- CONFIG_KCOV=y (upstream)
- based on compiler instrumentation (gcc6+)
- compiler inserts runtime callbacks into every basic block
- coverage per-thread per-syscall

Syscall Descriptions

Declarative description of all syscalls:

```
open(file filename, flags flags[open_flags],
      mode flags[open_mode]) fd
read(fd fd, buf buffer[out], size len[buf])
close(fd fd)

open_flags = O_RDONLY, O_WRONLY, O_RDWR, O_APPEND ...
```

Programs

These descriptions allows to **generate** and **mutate** "programs" in the following form:

```
mmap(&(0x7f0000000000), (0x1000), 0x3, 0x32, -1, 0)
r0 = open(&(0x7f0000000000)="./file0", 0x3, 0x9)
read(r0, &(0x7f0000000000), 42)
close(r0)
```

Algorithm

```
Start with empty corpus of programs;
while (true) {
    Generate a new program, or
        choose an existing program from corpus and mutate it;
    Interpret the program, collect coverage from every syscall;
    if (program covers code that wasn't covered previously)
        minimize the program and add it to corpus;
}
```

Operational Side

Ideally...

Reality...

Operation of a typical kernel fuzzer:

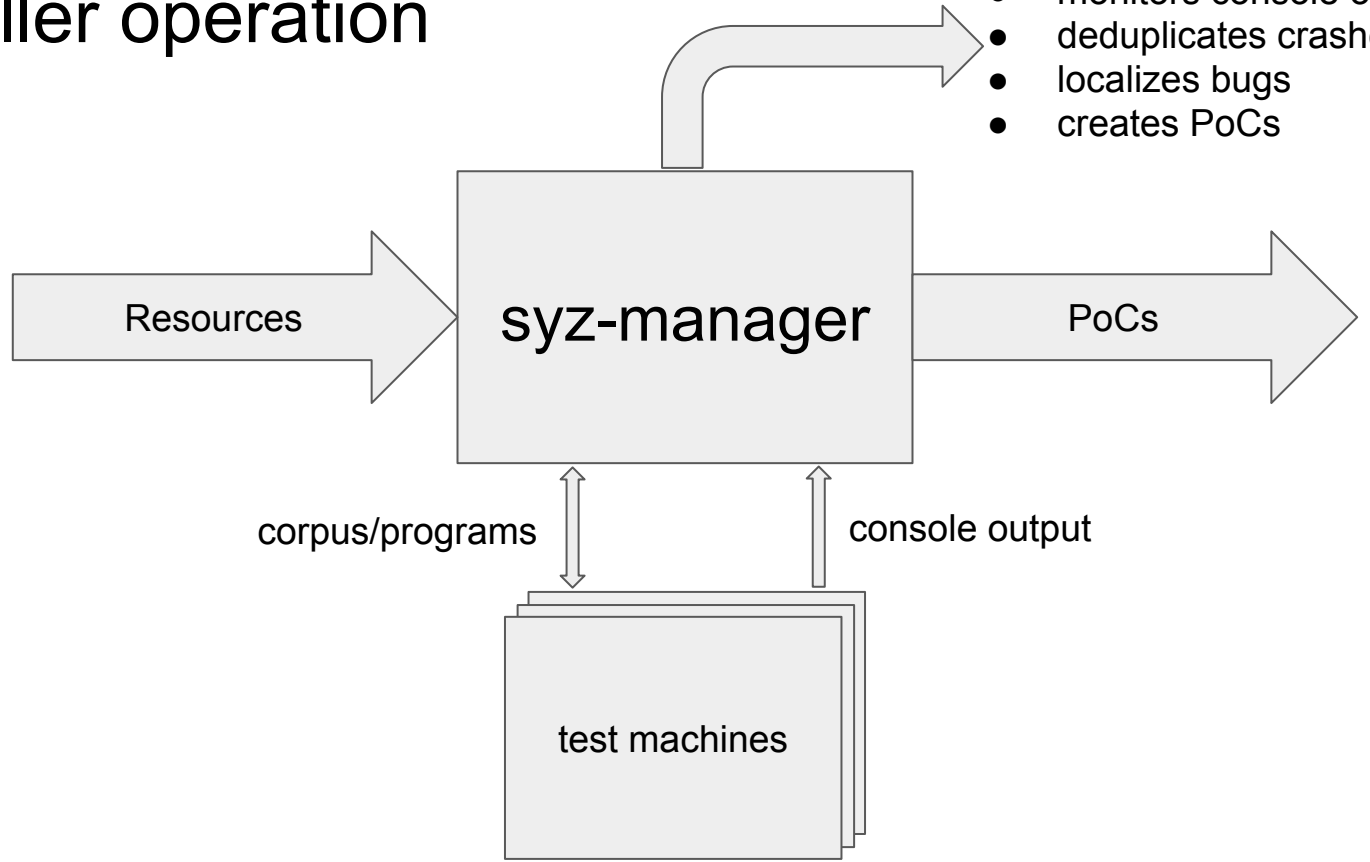
- **manually** create a bunch of VMs
- **manually** copy and start the binary
- **manually** monitor console output
- **manually** deduplicate crashes
- **manually** localize and reproduce
- **manually** restart the crashed VM

Works only if you want to find a handful of crashes. Otherwise -- full time job.

Worse for physical devices: you also need to stand there and press power button.

syzkaller operation

- manages test machines
- monitors console output
- deduplicates crashes
- localizes bugs
- creates PoCs



Test Machines

Currently supports:

- QEMU
- GCE (Google Compute Engine)
- Android devices (with serial cable/Suzy-Q)
- ODROID boards

The list is extensible, to support a new type need to:

- get console output (to grep for crashes)
- reboot/recreate (to repair after a crash)
- copy/run binary (think of scp/ssh)

Tutorial

Setting up syzkaller

- build kernel in a slightly special way
- build syzkaller
- write config file
- run syzkaller

Kernel Build

- Fresh compiler
  - gcc 7.0+ is perfect
  - gcc 6.0+ will work
  - gcc 5.0+ will work with sancov plugin
- CONFIG_KCOV for code coverage
- CONFIG_DEBUG_INFO for report symbolization
- CONFIG_KASAN for KASAN
- As many debug configs as possible (to get more bugs)
  - CONFIG_LOCKDEP
  - CONFIG_DEBUG_VM
  - CONFIG_DEBUG_ATOMIC_SLEEP

syzkaller Build

syzkaller is written in Go

- download Go 1.8.1 from golang.org/dl
- unpack it into ~/go1.8
- export PATH=~/go1.8/bin:$PATH
- go get -d github.com/google/syzkaller/...
- cd go/src/github.com/google/syzkaller
- make

syzkaller config for QEMU

```
{
"http": "localhost:12345",
"workdir": "./workdir",
"syzkaller": ".",
"kernel": "/linux/arch/x86/boot/bzImage",
"vmlinux": "/linux/vmlinux",
"image": "/image/wheezy.img",
"sshkey": "/image/id_rsa",
"type": "qemu",
"count": 4,
"procs": 8,
"cpu": 2,
"mem": 2048,
"sandbox": "none"
}
```

 # local address for web UI
 # shows crashes, coverage, stats, etc

syzkaller config for QEMU

```
{
"http": "localhost:12345",
"workdir": "./workdir",
"syzkaller": ".",
"kernel": "/linux/arch/x86/boot/bzImage",
"vmlinux": "/linux/vmlinux",
"image": "/image/wheezy.img",
"sshkey": "/image/id_rsa",
"type": "qemu",
"count": 4,
"procs": 8,
"cpu": 2,
"mem": 2048,
"sandbox": "none"
}
```

 # local dir to save corpus and crashes
 # syzkaller checkout dir

syzkaller config for QEMU

```
{
"http": "localhost:12345",
"workdir": "./workdir",
"syzkaller": ".",
"kernel": "/linux/arch/x86/boot/bzImage",   # compressed kernel
"vmlinux": "/linux/vmlinux",                # kernel object file (for symbolization)
"image": "/image/wheezy.img",
"sshkey": "/image/id_rsa",
"type": "qemu",
"count": 4,
"procs": 8,
"cpu": 2,
"mem": 2048,
"sandbox": "none"
}
```

syzkaller config for QEMU

```
{
"http": "localhost:12345",
"workdir": "./workdir",
"syzkaller": ".",
"kernel": "/linux/arch/x86/boot/bzImage",
"vmlinux": "/linux/vmlinux",
"image": "/image/wheezy.img",
"sshkey": "/image/id_rsa",
"type": "qemu",
"count": 4,
"procs": 8,
"cpu": 2,
"mem": 2048,
"sandbox": "none"
}
```

 # disk image and root ssh key
 # can be created with:
 # ./tools/create-image.sh

syzkaller config for QEMU

```
{
"http": "localhost:12345",
"workdir": "./workdir",
"syzkaller": ".",
"kernel": "/linux/arch/x86/boot/bzImage",
"vmlinux": "/linux/vmlinux",
"image": "/image/wheezy.img",
"sshkey": "/image/id_rsa",
"type": "qemu",
"count": 4,
"procs": 8,
"cpu": 2,
"mem": 2048,
"sandbox": "none"
}
```

 # test machine type
 # one of "qemu", "gce", "adb", "odroid"
 # other types need slightly different config

syzkaller config for QEMU

```
{
"http": "localhost:12345",
"workdir": "./workdir",
"syzkaller": ".",
"kernel": "/linux/arch/x86/boot/bzImage",
"vmlinux": "/linux/vmlinux",
"image": "/image/wheezy.img",
"sshkey": "/image/id_rsa",
"type": "qemu",
"count": 4,
"procs": 8,
"cpu": 2,
"mem": 2048,
"sandbox": "none"
}
```

 # number of VMs to use
 # number of parallel tests per VM
 # CPUs per VM
 # RAM per VM

syzkaller config for QEMU

```
{
"http": "localhost:12345",
"workdir": "./workdir",
"syzkaller": ".",
"kernel": "/linux/arch/x86/boot/bzImage",
"vmlinux": "/linux/vmlinux",
"image": "/image/wheezy.img",
"sshkey": "/image/id_rsa",
"type": "qemu",
"count": 4,
"procs": 8,
"cpu": 2,
"mem": 2048,
"sandbox": "none"
}
```

 # test sandboxing mode
 # "none": run under root as is
 # "setuid": do setuid(nobody)
 # "namespace": run in namespaces

syzkaller config for Android

```
{
"http": "localhost:12345",
"workdir": "./workdir",
"syzkaller": ".",
"kernel": "/linux/arch/x86/boot/bzImage",
"vmlinux": "/linux/vmlinux",
"image": "/image/wheezy.img",
"sshkey": "/image/id_rsa",
"type": "qemu",
"count": 4,
"procs": 8,
"cpu": 2,
"mem": 2048,
"sandbox": "none",
"type": "adb",
"devices": ["ADBDEVID1", "ADBDEVID2", "ADBDEVID3"]
}
```

Ready to go

```
$ bin/syz-manager -config my.cfg
```

Extending syzkaller

Steps to extend syzkaller

Let's say we want to fuzz **/dev/ion** device on Android.

1. Create new file for descriptions: `sys/ion.txt`
2. Write descriptions for the new system calls
3. Rebuild
4. Run

sys/ion.txt: includes

```
# includes are required to extract
# values of literal constants used in the descriptions.

# for ioctl-related constants
include <asm/ioctl.h>

# for ion-related constants
include <drivers/staging/android/uapi/ion.h>
```

sys/ion.txt: resources

```
# resources allow syzkaller to pass output of one syscall
# as an input to another syscall.

# these 2 are "derived" from fd
resource fd_ion[fd]
resource fd_ion_generic[fd]

# this is "derived" from int32
resource ion_handle[int32]
```

sys/ion.txt: system calls

```
open$ion(file ptr[in, string["/dev/ion"]],
    flags flags[open_flags], mode const[0]) fd_ion

ioctl$ION_IOC_ALLOC(fd fd_ion, cmd const[ION_IOC_ALLOC],
    arg ptr[inout, ion_allocation_data])
ioctl$ION_IOC_FREE(fd fd_ion, cmd const[ION_IOC_FREE],
    arg ptr[in, ion_handle_data])
ioctl$ION_IOC_MAP(fd fd_ion, cmd const[ION_IOC_MAP],
    arg ptr[inout, ion_fd_data])
... few more of them ...
```

sys/ion.txt: structs

```
ion_allocation_data {
  len      intptr
  align    intptr
  heapid   int32
  flags    int32
  handle   ion_handle
}

ion_fd_data {
  handle ion_handle
  fd     fd_ion_generic
}
```

Rebuilding

```
--- a/extract.sh
+++ b/extract.sh
-ANDROID_FILES="sys/tlk_device.txt"
+ANDROID_FILES="sys/tlk_device.txt sys/ion.txt"
```

```
# updates .const files
$ make extract ANDROID=/path/to/android/kernel/source

# rebuilds binaries
$ make
```

sys/ion_arm64.const

```
# AUTOGENERATED FILE
ION_IOC_ALLOC = 3223341312
ION_IOC_CUSTOM = 3222292742
ION_IOC_FREE = 3221506305
ION_IOC_IMPORT = 3221768453
ION_IOC_MAP = 3221768450
ION_IOC_SHARE = 3221768452
ION_IOC_SYNC = 3221768455
__NR_ioctl = 29
```

Ready to go

```
$ bin/syz-manager -config my.cfg
```

~1/4 of the bugs we found

tty: fix port buffer locking
kvm: warning in kvm_load_guest_fpu
drivers/scsi: GPF in sg_read
net/ipv4: use-after-free in ip_mc_drop_socket
net/ipv6: GPF in rt6_device_match
x86: warning: kernel stack regs has bad 'bp' value
net/key: slab-out-of-bounds in pfkey_compile_policy
net/ipv6: warning in inet6_ifa_finish_destroy
net/ipv6: use-after-free in __call_rcu/in6_dev_finish_destroy_rcu
net/ipv6: slab-out-of-bounds in ip6_tnl_xmit
net/rose: null-ptr-deref in rose_route_frame
time: hang due to timer_create/timer_settime
net/core: BUG in unregister_netdevice_many
net/xfrm: stack-out-of-bounds in xfrm_state_find
net/bonding: stack-out-of-bounds in bond_enslave
net: ipv6: RTF_PCPU should not be settable from userspace
fs/notify/inotify: slab-out-of-bounds write in strcpy
net/ipv6: slab-out-of-bounds read in seg6_validate_srh
kernel BUG at mm/hugetlb.c:742!
net/key: slab-out-of-bounds in parse_ipsecrequests
net/ipv4: use-after-free in ipv4_datagram_support_cmsg
net/ipv4: use-after-free in ip_queue_xmit
net: use-after-free in __ns_get_path
net/ipv4: use-after-free in ip_check_mc_rcu
net/ipv6: use-after-free in ipv6_sock_ac_close
net/ipv4: use-after-free in ipv4_mtu
net/dccp: BUG in tfrc_rx_hist_sample_rtt
net/sctp: list double add warning in sctp_endpoint_add_asoc
kvm: use-after-free in srcu_reschedule
ata: WARNING in ata_bmdma_qc_issue
net/sched: GPF in qdisc_hash_add
sg: random memory corruptions
fs: GPF in deactivate_locked_super
loop: WARNING in sysfs_remove_group
lib, fs, cgroup: WARNING in percpu_ref_kill_and_confirm
ata: WARNING in ata_qc_issue
security, hugetlbfs: write to user memory in hugetlbfs_destroy_inode
netlink: NULL timer crash
kvm: use-after-free function call in kvm_io_bus_destroy
sound: use-after-free in snd_seq_cell_alloc
usb: use-after-free write in usb_hcd_link_urb_to_ep
net/kcm: double free of kcm inode
crypto: out-of-bounds write in pre_crypt
security: double-free in superblock_doinit
kvm: WARNING in kvm_apic_accept_events
tcp: fix potential double free issue for fastopen_req
net/udp: slab-out-of-bounds Read in udp_recvmsg
net: deadlock between ip_expire/sch_direct_xmit
srcu: BUG in __synchronize_srcu
net/sctp: recursive locking in sctp_do_peeloff

kvm: WARNING in vmx_handle_exit
futex: use-after-free in futex_wait_requeue_pi
kvm/arm64: use-after-free in kvm_vm_ioctl/vmacache_update
kvm/arm64: use-after-free in kvm_unmap_hva_handler/unmap_stage2_pmds
local privilege escalation flaw in n_hdlc CVE-2017-2636
netlink: GPF in netlink_unicast
perf: use-after-free in perf_release
net/ipv6: null-ptr-deref in ip6mr_sk_done
bpf: kernel NULL pointer dereference in map_get_next_key
crypto: deadlock between crypto_alg_sem/rtnl_mutex/genl_mutex
kvm: use-after-free in vmx_check_nested_events/vmcs12_guest_cr0
sound: another deadlock in snd_seq_pool_done
rcu: WARNING in rcu_seq_end
fs: use-after-free in path_lookupat
ucount: use-after-free read in inc_ucount & dec_ucount
net/ipv4: division by 0 in tcp_select_window
net: heap out-of-bounds in fib6_clean_node/rt6_fill_node/fib6_age/fib6_prune_clone
mm: use-after-free in zap_page_range
net/kcm: use-after-free in kcm_wq
idr: use-after-free write in ida_get_new_above
sg: stack out-of-bounds write in sg_write CVE-2017-7187
cgroup: WARNING in cgroup_kill_sb
net/rds: use-after-free in rds_find_bound/memcmp
net: sleeping function called from invalid context in net_enable_timestamp
net: use-after-free in neigh_timer_handler/sock_wfree
net/sctp: use-after-free in sctp_association_put
fs: use-after-free in userfaultfd_exit
net/ipv4: inconsistent lock state in tcp_conn_request/inet_ehash_insert
net/ipv4: suspicious RCU usage in ip_ra_control
net/ipv4: deadlock in ip_ra_control
net/dccp: dccp_create_openreq_child freed held lock
nested_vmx_merge_msr_bitmap
ipc: use-after-free in shm_get_unmapped_area
sounds: deadlocked processed in snd_seq_pool_done
net/atm: vcc_sendmsg calls kmem_cache_alloc in non-blocking context
ata: WARNING in ata_sff_qc_issue
net/rds: use-after-free in inet_create
mm: fault in __do_fault
kvm: WARNING in nested_vmx_vmexit
net: GPF in rt6_nexthop_info
sound: spinlock lockup in snd_timer_user_tinterrupt
mm: GPF in bdi_put
net/sctp: use-after-free in sctp_hash_transport
net/bridge: warning in br_fdb_find
net/ipv6: null-ptr-deref in ip6_route_del/lock_acquire
net: possible deadlock in skb_queue_tail

DCCP double-free vulnerability (local root) CVE-2017-6074
net: warning in inet_sock_destruct
net/pptp: use-after-free in dst_release
net/udp: slab-out-of-bounds in udp_recvmsg/do_csum CVE-2017-6347
WARNING in skb_warn_bad_offload
tty: panic in tty_ldisc_restore
net: BUG in __skb_gso_segment
net/dccp: use-after-free in dccp_feat_activate_values
net/kcm: GPF in kcm_sendmsg
net/xfrm: stack out-of-bounds in xfrm_flowi_sport
net/llc: BUG in llc_sap_state_process/skb_set_owner_r CVE-2017-6345
net/llc: bug in llc_pdu_init_as_xid_cmd/skb_over_panic
net/packet: use-after-free in packet_rcv_fanout
net: SOFTIRQ-safe -> SOFTIRQ-unsafe lock order detected in skb_array_produce
net/ipv4: null-ptr-deref in udp_rmem_release/sk_memory_allocated_sub
net/sctp: null-ptr-deref in sctp_put_port/sctp_endpoint_destroy
net/ipv4: warning in nf_nat_ipv4_fn
net/ipv6: double free in ipip6_dev_free
sound: use-after-free in snd_seq_queue_alloc
loop: divide error in transfer_xor
net/xfrm: use of uninit spinlock in xfrm_policy_flush
[illegible]m: double free in [illegible]vb_b[illegible]_init
packet: round up linear to header len
net/icmp: null-ptr-deref in ping_v4_push_pending_frames
net/kcm: WARNING in kcm_write_msgs
tcp: avoid infinite loop in tcp_splice_read() CVE-2017-6214
tun: read vnet_hdr_sz once
macvtap: read vnet_hdr_size once
udp: properly cope with csum errors
ipv6: tcp: add a missing tcp_v6_restore_cb()
ip6_gre: fix ip6gre_err() invalid reads CVE-2017-5897
ipv4: keep skb->dst around in presence of IP options CVE-2017-5970
net: use a work queue to defer net_disable_timestamp() work
netlabel: out of bound access in cipso_v4_validate()
ipv6: pointer math error in ip6_tnl_parse_tlv_enc_lim()
net: heap out-of-bounds in ip6_fragment
tcp: fix 0 divide in __tcp_select_window()
keys: GPF in request_key
net/tcp: warning in tcp_try_coalesce/skb_try_coalesce
crypto: NULL deref in sha512_mb_mgr_get_comp_job_avx2
sound: unable to handle kernel paging request snd_seq_prioq_cell_out
scsi: BUG in scsi_init_io
mm: sleeping function called from invalid context shmem_undo_range
timerfd: use-after-free in timerfd_remove_cancel
scsi: use-after-free in sg_start_req

mm: deadlock between get_online_cpus/pcpu_alloc
BUG at net/sctp/socket.c:7425
kvm: use-after-free in irq_bypass_register_consumer
net: suspicious RCU usage in nf_hook
kvm: fix page struct leak in handle_vmon CVE-2017-2596
ipv6: fix ip6_tnl_parse_tlv_enc_lim()
kvm: WARNING in mmu_spte_clear_track_bits
perf: use-after-free in perf_event_for_each
net: use-after-free in tw_timer_handler
namespace: deadlock in dec_pid_namespaces
sctp: kernel memory overwrite attempt detected in sctp_getsockopt_assoc_stats
kvm: deadlock in kvm_vgic_map_resources
net/atm: warning in alloc_tx/__might_sleep
net/ipv6: use-after-free in sock_wfree
kvm: kvm: BUG in loaded_vmcs_init
kvm: NULL deref in vcpu_enter_guest
kvm: use-after-free in complete_emulated_mmio CVE-2017-2584
kvm: BUG in kvm_unload_vcpu_mmu
x86: warning in unwind_get_return_address
ipc: BUG: sem_unlock unlocks non-locked lock
kvm: WARNING in mmu_spte_clear_track_bits
sctp: suspicious rcu_dereference_check() usage in sctp_epaddr_lookup_transport
usb/core: warning in usb_create_ep_devs/sysfs_create_dir_ns
usb/gadget: warning in ep_write_iter/__alloc_pages_nodemask
kvm: use-after-free in process_srcu
kvm: assorted bugs after OOMs
kvm: deadlock between kvm_io_bus_register_dev/kvm_hv_set_msr_common
netlink: GPF in netlink_dump
fs, net: deadlock between bind/splice on af_unix
usb/gadget: slab-out-of-bounds write in dev_config
usb/gadget: warning in dummy_free_request
usb/gadget: use-after-free in gadgetfs_setup
usb/gadget: GPF in usb_gadget_unregister_driver
net: use-after-free in worker_thread
net: signed overflows in SO_{SND|RCV}BUFFORCE sockopts CVE-2016-9793 CVE-2012-6704
usb/gadget: warning in dev_config/memdup_user
net/can: warning in raw_setsockopt/__alloc_pages_slowpath
net/ipv6: null-ptr-deref in ip6_rt_cache_alloc
net/dccp: use-after-free in dccp_invalid_packet
net/sctp: vmalloc allocation failure in sctp_setsockopt/xt_alloc_table_info
net: BUG in unix_notinflight
net: GPF in eth_header CVE-2016-9755
net: deadlock on genl_mutex
net: GPF in rt6_get_cookie

Thank you!

Q&A

github.com/google/syzkaller
syzkaller@googlegroups.com

Dmitry Vyukov, dvyukov@

Links

https://github.com/google/syzkaller

https://www.kernel.org/doc/html/latest/dev-tools/kasan.html

https://github.com/google/kmsan

https://github.com/google/ktsan

Backup

Our Team: Dynamic Testing Tools

- User-space tools: ASAN, MSAN, TSAN
- Kernel tools: KASAN, KMSAN, KTSAN
- Hardening: CFI, SafeStack
- Fuzzing: LibFuzzer, syzkaller, OSS-Fuzz

KMSAN: What one might think it does

```
int a;
b = c + a;                  // report reading of uninit a
```

or:

```
int a = x;
copy_to_user(p, &a, 4); // don't report since a was inited
```

This is useless: both false positives and false negatives!

KMSAN: What is does

```
int a;
if (flag)
    a = 1;                      // initialize
b = c + a;                      // not a "use"
if (flag)
    copy_to_user(p, &b, 4); // use: don't report
```

or:

```
int x;                          // not initialized
int a = x;                      // still not initialized
copy_to_user(p, &a, 4);     // use: report
```

Coverage-guiding in action

```
if (input[0] == '{') {
     if (input[1] == 'i' && input[2] == 'f') {
          if (input[3] == '(') {
               input[input[4]] = input[5]; // potential OOB write
          }
     }
}
```

Requires **"{if("** input to crash, ~2^32 guesses to crack when blind.

Coverage-guiding:
Guess **"{"** in ~2^8, add to corpus.
Guess **"{i"** in ~2^8, add to corpus.
Guess **"{if"** in ~2^8, add to corpus.
Guess **"{if("** in ~2^8, add to corpus.
Total: ~2^10 guesses.

Syscall Descriptions: discriminated syscalls

Can describe families of discriminated syscalls:

```
fcntl$setflags(fd fd, cmd const[F_SETFD], flags flags[fcntl_flags])
fcntl$setstatus(fd fd, cmd const[F_SETFL], flags flags[fcntl_status])
fcntl$getown(fd fd, cmd const[F_GETOWN]) pid
fcntl$setown(fd fd, cmd const[F_SETOWN], pid pid)
fcntl$setsig(fd fd, cmd const[F_SETSIG], sig signalno)
fcntl$setlease(fd fd, cmd const[F_SETLEASE], typ flags[flock_type])
```

Syscall Descriptions: structs/unions

```
fcntl$getownex(fd fd, cmd const[F_GETOWN_EX], arg  ptr[out, f_owner_ex])

# struct:
f_owner_ex {
    type    flags[f_owner_type, int32]
    pid     pid
}

# union:
tun_buffer [
    pi      tun_pi
    hdr     virtio_net_hdr
] [varlen]
```

Syscall Descriptions: resources

```
resource fd_bpf_map[fd]
resource fd_bpf_prog[fd]

bpf$MAP_CREATE(cmd const[BPF_MAP_CREATE], ...) fd_bpf_map

bpf_map_lookup_arg {
    map    fd_bpf_map
    key    buffer[in]
    val    buffer[out]
}
```