Dynamic program analysis for fun and profit

Dmitry Vyukov, Principal Software Engineer, Google
Agenda:

- Dynamic analysis
- Overview of dynamic tools
- KASAN
Dynamic Tools Team

- **Bug detection (user-space/kernel):**
  - ASAN
  - MSAN
  - TSAN (C++, Go, Java)
  - KCSAN
  - LSAN
  - UBSAN

- **Production hardening:**
  - CFI
  - SafeStack
  - ShadowCallStack
  - HWASAN
  - Memory tagging (MTE)
  - GWP-ASan
  - KFENCE

- **Bug provocation:**
  - LibFuzzer (C++ [Go, Rust])
  - go-fuzz (Go)
  - syzkaller (kernels)

- **Misc:**
  - OSS-Fuzz
  - syzbot
  - SanitizerCoverage
  - KCOV
  - DFSAN
Why?

- **Bugs** == security issues
- **Bugs** == stability issues
- **Bugs** == low quality
- **Bugs** == wasted time
- **Bugs** == moving slow

**Dynamic analysis** == cost-effective way to find **bugs**
Dynamic program analysis -
analysis of the properties of a running program

Properties:
• bugs
• performance
• code coverage
• call graph
• data flow
Dynamic program analysis -
analysis of the properties of a running program
(properties that hold on a single execution)

Static program analysis -
analysis of the properties of program code
(properties that hold on all executions)
True Positive - report a real bug

False Positive - report not a bug
True Positives

No False Positives
<table>
<thead>
<tr>
<th></th>
<th>Static Analysis</th>
<th>Dynamic Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positives</td>
<td>+</td>
<td>—</td>
</tr>
<tr>
<td>No False Positives</td>
<td>—</td>
<td>+</td>
</tr>
</tbody>
</table>
void func() {
  char* p = malloc(10);
  p[20] = 1; // OOB (out-of-bounds)
}

Static Analysis
void func(int index) {
    char* p = malloc(10);
    p[index] = 1; // OOB?
}

char* str = &buffer[offset];
int index = atoi(str);
func(index);
void func(int index) {
    char* p = malloc(10); // memorize size of *p
    p[index] = 1; // if index >= size *p
    // report bug
}

• doesn't matter how index was computed (atoi, unions, ...)
• never report a false positive
• always report the bug
• caveat: only on test that trigger the bug
<table>
<thead>
<tr>
<th>Tool</th>
<th>Warnings</th>
<th>Time (CPU/days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clang --analyze*</td>
<td>10204</td>
<td>1.28</td>
</tr>
<tr>
<td>gcc -fanalyze*</td>
<td>461</td>
<td>0.24</td>
</tr>
<tr>
<td>Coccinelle</td>
<td>4526</td>
<td>1.5</td>
</tr>
<tr>
<td>Smatch*</td>
<td>405</td>
<td>1.8</td>
</tr>
</tbody>
</table>

* only analyze 1 configuration out of lots (12316 configs + different arches)
Static
- simpler, local bugs
- more code coverage
- faster, deterministic feedback

Dynamic
- more complex bugs
- no false positives
- simper usage model
Getting coverage:

• tests (unit, system)
• fuzzing (syzkaller)
• development
• pre-production, mirror servers, dogfood clients
• production
Kernel tests:

- **KUnit**
- **kselftests** (tools/testing/selftests/*)
- out-of-tree suites:
  - Linux Test Project
  - xfstests
  - v4l2-compliance
  - ...
DIY Tools
(“Do It Yourself”)

LF Live MENTORSHIP SERIES
CONFIG_DEBUG_LIST=y

struct list_head {
    struct list_head *next;
    struct list_head *prev;
};

void __list_del_entry_valid(struct list_head *entry) {
    #ifdef CONFIG_DEBUG_LIST
        BUG_ON(entry->next == LIST_POISON1);
        BUG_ON(entry->prev == LIST_POISON2);
        BUG_ON(entry->prev->next != entry);
        BUG_ON(entry->next->prev != entry);
    #endif
}
list_del corruption, ffff88800771fe58->next is LIST_POISON1 (dead000000000100)
-------------[ cut here ]-------------
kernel BUG at lib/list_debug.c:45!
CPU: 0 PID: 1 Comm: swapper/0 Not tainted 5.11.0-rc7+ #74
RIP: 0010:__list_del_entry_valid+0xf/0x47 lib/list_debug.c:45
Call Trace:
__list_del_entry include/linux/list.h:132 [inline]
list_del include/linux/list.h:146 [inline]
test_init+0x79/0x115 kernel/test.c:1040
do_one_initcall+0x69/0x290 init/main.c:1223
do_initcall_level init/main.c:1296 [inline]
do_initcalls init/main.c:1312 [inline]
do_basic_setup init/main.c:1332 [inline]
kernel_init_freeable+0x1cd/0x249 init/main.c:1533
kernel_init+0x10/0x1b1 init/main.c:1421
ret_from_fork+0x1f/0x30 arch/x86/entry/entry_64.S:296
---[ end trace 1526104c6066be33 ]---
char buf[10];
memset(buf, 0, size);

void *memset(void *p, int c, size_t size)
{
    size_t p_size = __builtin_object_size(p);
    if (p_size < size)
        __write_overflow();
    return __underlying_memset(p, c, size);
}
BUG_ON(condition)
WARN_ON(condition)

void do_group_exit(int exit_code)
{
  BUG_ON(exit_code & 0x80); /* core dumps don't get here */
  ...
}
- DEBUG_LIST
- DEBUG_PLIST
- FORTIFY_SOURCE
- DEBUG_KOBJECT
- SCHED_STACK_END_CHECK
- HARDENED_USERCOPY
- HARDENED_USERCOPY_FALLBACK
- LOCKUP_DETECTOR
- SOFTLOCKUP_DETECTOR
- DETECT_HUNG_TASK
- WQ_WATCHDOG
- HARDLOCKUP_DETECTOR
- DEBUG_SG
- LOCKDEP
- PROVE_LOCKING
- DEBUG_ATOMIC_SLEEP
- PROVE_RCU
- RCU_EQS_DEBUG
- DEBUG_LOCK_ALLOC
- DEBUG_RT_MUTEXES
- DEBUG_SPINLOCK
- DEBUG_MUTEXES
- DEBUG_WW_MUTEX_SLOWPATH
- DEBUG_RWSEMS
- SND_DEBUG
- SND_PCM_XRUN_DEBUG
• DEBUG_OBJECTS
• DEBUG_OBJECTS_ENABLE_DEFAULT
• DEBUG_OBJECTS_FREE
• DEBUG_OBJECTS_PERCPU_COUNTER
• DEBUG_OBJECTS_RCU_HEAD
• DEBUG_OBJECTS_TIMERS
• DEBUG_OBJECTS_WORK
• DEBUG_PREEMPT
• DEBUG_DEVRES
• DEBUG_NOTIFIERS
• DEBUG_CREDENTIALS
• UBSAN_BOUNDS
• UBSAN_SHIFT
• DEBUG_VM
• DEBUG_VM_RB
• DEBUG_VM_VMACACHE
• DEBUG_VM_PGFLAGS
• DEBUG_VM_PGTABLE
• DEBUG_VIRTUAL
• DEBUG_KMAP_LOCAL_FORCE_MAP
• DEBUG_MEMORY_INIT
• PAGE_POISONING
• RING_BUFFER_VALIDATE_TIME_DELTAS
• DYNAMIC_DEBUG
• SND_CTL_VALIDATION
• DEBUG_PER_CPU_MAPS
- DEBUG_KMEMLEAK
- FAULT_INJECTION
- FAILSLAB
- FAIL_PAGE_ALLOC
- FAIL_MAKE_REQUEST
- FAIL_IO_TIMEOUT
- FAIL_FUTEX
- FAULT_INJECTION_DEBUG_FS
- FAULT_INJECTION_USERCOPY
- DEBUG_INFO
- DEBUG_BUGVERBOSE
- PRINTK_CALLER
- PANIC_ON_OOPS
- BUG_ON_DATA_CORRUPTION
- BOOTPARAM_HARDLOCKUP_PANIC
- BOOTPARAM_HUNG_TASK_PANIC
- BOOTPARAM_SOFTLOCKUP_PANIC
- panic_on_warn (command line)
$ scripts/decode_stacktrace.sh vmlinux < crash.log

do_one_initcall (init/main.c:1223)
KASAN

(KernelAddressSANitizer)
Pillars:

- no false positives
- work out of the box
- informative reports
- low overhead
KASAN:

- Out-Of-Bounds
- Use-After-Free
- Heap, stack, globals
- CONFIG_KASAN=y
p = kmalloc(10);
p[20] = 1; // out-of-bounds

kfree(p);
p[0] = 1; // use-after-free
BUG: KASAN: **use-after-free** in remove_wait_queue
Write of size 8 by task syzkaller/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
...
SYSC_write fs/read_write.c:585
Kernel Memory  Shadow

8 aligned bytes  1 byte
Kernel Memory

Shadow

- good to access byte
- bad byte (out-of-bounds, freed)
Shadow = Addr/8 + Offset

[see Documentation/x86/x86_64/mm.rst]
Red-zones around heap objects:

*compiler arranges red-zones for stack/globals*
Quarantine for heap objects:

- `kfree()`
- `kmalloc()`
- LIFO
- `kfree()`
- `kmalloc()`
- `kfree()`
- `kmalloc()`
- `kfree()`
- `kmalloc()`

Quarantine (delayed reuse)
Compiler instrumentation:

```c
shadow = p >> 3 + 0xdfffffc00000000000;
if (*shadow)
    kasan_report8(p);
*p = 1;  // 8 bytes
```
Compiler instrumentation:

shadow = p >> 3 + 0xdffffc000000000000;
if (*shadow && *shadow <= ((p & 7) + N - 1))
  kasan_reportN(p);
*p = 1; // N = 1, 2, 4 bytes
KASAN summary:

- shadow memory marks good/bad bytes
- red-zones (out-of-bounds)
- quarantine (use-after-free)
- shadow checks before memory accesses
• no false positives
  – by design
• work out of the box
  – just enable the config
• informative reports
  – alloc/free stacks
  – heap/stack/global object description
  – last call_rcu(), queue_work() stacks
• low overhead
  – ~2x slowdown; ~2x memory overhead
How good is it?
<table>
<thead>
<tr>
<th>Title</th>
<th>Repro</th>
<th>Cause bisect</th>
<th>Fix bisect</th>
<th>Count</th>
<th>Last</th>
<th>Reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUG: soft lockup in ie80211_tasklet_handler</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>15h27m</td>
<td>15h26m</td>
</tr>
<tr>
<td>WARNING in ie802154_get_llsec_params</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>19h46m</td>
<td></td>
</tr>
<tr>
<td>possible deadlock in io_link_timeout_in</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>19h46m</td>
<td></td>
</tr>
<tr>
<td>UBSAN: shift-out-of-bounds in detach_tasks</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>19h46m</td>
<td></td>
</tr>
<tr>
<td>WARNING in hid_alloc_report_buf</td>
<td></td>
<td>unreliable</td>
<td></td>
<td>2</td>
<td>1d01h</td>
<td></td>
</tr>
<tr>
<td>WARNING in ie802154_del_scelvel</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1d23h</td>
<td></td>
</tr>
<tr>
<td>memory leak in __pskb_copy_fclone</td>
<td>C</td>
<td></td>
<td></td>
<td>1</td>
<td>1d23h</td>
<td></td>
</tr>
<tr>
<td>memory leak in do_seccomp(2)</td>
<td>C</td>
<td></td>
<td></td>
<td>1</td>
<td>1d23h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in ie802154_llsec_parse_key_id</td>
<td>C</td>
<td>inconclusive</td>
<td></td>
<td>4</td>
<td>1d23h</td>
<td></td>
</tr>
<tr>
<td>KMSAN: kernel-infoleak in compat_drm_wait_vblank</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>2d08h</td>
<td></td>
</tr>
<tr>
<td>WARNING in nbd_dev_add</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2d08h</td>
<td></td>
</tr>
<tr>
<td>memory leak in iget_locked</td>
<td>syz</td>
<td></td>
<td></td>
<td>1</td>
<td>2d08h</td>
<td></td>
</tr>
<tr>
<td>memory leak in con_do_clear_unimap</td>
<td>C</td>
<td></td>
<td></td>
<td>1</td>
<td>2d08h</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in nbd_release</td>
<td>C</td>
<td>inconclusive</td>
<td></td>
<td>3</td>
<td>2d08h</td>
<td></td>
</tr>
<tr>
<td>general protection fault in nbd_disconnect_and_put</td>
<td>C</td>
<td>unreliable</td>
<td></td>
<td>10</td>
<td>2d08h</td>
<td></td>
</tr>
<tr>
<td>UBSAN: shift-out-of-bounds in nl802154_new_interface</td>
<td>C</td>
<td>inconclusive</td>
<td></td>
<td>17</td>
<td>2d08h</td>
<td></td>
</tr>
<tr>
<td>INFO: rcu detected stall in __hrtimer_run_queues</td>
<td>C</td>
<td></td>
<td></td>
<td>18</td>
<td>3d12h</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Write in j1939_can_recv</td>
<td></td>
<td>inconclusive</td>
<td></td>
<td>1</td>
<td>3d23h</td>
<td></td>
</tr>
<tr>
<td>WARNING in netbl_cipsov4_add</td>
<td>C</td>
<td></td>
<td></td>
<td>105</td>
<td>4d02h</td>
<td></td>
</tr>
<tr>
<td>KASAN: out-of-bounds Read in leaf_paste_entries</td>
<td>C</td>
<td>inconclusive</td>
<td></td>
<td>2</td>
<td>4d03h</td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Read in ip6_pol_route(2)</td>
<td>C</td>
<td>done</td>
<td></td>
<td>2</td>
<td>4d23h</td>
<td></td>
</tr>
</tbody>
</table>
> 5000 bugs reported
  - KASAN: 1000
  - KMSAN: 375
  - KCSAN: 480
  - LOCKDEP: 170
  - WARNING+BUG: 1000
  - NULL deref: 500

> 3000 bugs fixed

> 4500 LTS backports

sources: [1], [2], [3], [4]
Dynamic tools are your friends!

- enable DEBUG_XXX, LOCKDEP, KASAN
- use during development
- insert BUG_ON / WARN_ON
- add/run tests
- scripts/decode_stacktrace.sh
Bug fix - perfect first contribution!

syzkaller.appspot.com

Linux kernel Bug Fixing LF Mentorship program
Contributions are welcome!

**bugzilla laundry list:**

- [206267] KASAN: missed checks in copy_to/from_user
- [206269] KASAN: missed checks in ioread/write8/16/32_rep
- [199341] KASAN: misses underflow in memmove
- [203967] KASAN: incorrect alloc/free stacks for alloc_pages
- [199055] KASAN: poison skb linear data tail
- ...
Sanitizers in user-space!

- `clang/gcc -fsanitize=address`
  - use-after-free, out-of-bounds
- `clang/gcc -fsanitize=thread`
  - data races
- `clang/gcc -fsanitize=memory`
  - uses of uninit values
- `clang/gcc -fsanitize=undefined`
  - overflows, alignment, truncations, ...
Thank you for joining us today!

We hope it will be helpful in your journey to learning more about effective and productive participation in open source projects. We will leave you with a few additional resources for your continued learning:

- The **LF Mentoring Program** is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- **Outreachy remote internships program** supports diversity in open source and free software.
- **Linux Foundation Training** offers a wide range of **free courses**, webinars, tutorials and publications to help you explore the open source technology landscape.
- **Linux Foundation Events** also provide educational content across a range of skill levels and topics, as well as the chance to meet others in the community, to collaborate, exchange ideas, expand job opportunities and more. You can find all events at [events.linuxfoundation.org](http://events.linuxfoundation.org).