Fuzzing the kernel

Andrew Turner

andrew@FreeBSD.org

Who am I?

- FreeBSD Committer (andrew@)
- Research Associate in the University of Cambridge
- Sometimes a Freelance Software Engineer

Sanitizers

Sanitizers

- A tool from the compiler to instrument code
- Add function calls on in interesting points in the code, e.g.:
  - Entry to a basic block
  - On comparison operations
  - Every memory access
- The compiler provides a runtime for userspace
- We need our own runtime in the kernel

Undefined Behaviour

KUBSAN – Undefined Behaviour Sanitizer

- Instruments code that may be undefined depending on input, e.g.
  - Misaligned or NULL pointer
  - Shift out of bounds

- From the NetBSD µUBsan
  - imported 3 August 2018
- Imported into FreeBSD 6 November 2018
  - Off by default (large increase in kernel file size)
- Imported into OpenBSD 18 March 2019

KUBSAN Reports

- Misaligned memory access:
  - UBSan: Undefined Behavior in …/sys/vm/uma_core.c:1746:8, member access within misaligned address 0xfffff8087ffde7c0 for type 'struct uma_zone' which requires 128 byte alignment
- NULL pointer dereference:
  - UBSan: Undefined Behavior in …/sys/contrib/ck/src/ck_epoch.c:143:1, member access within null pointer of type 'struct ck_epoch_record’
- Shift out of bounds:
  - UBSan: Undefined Behavior in …/sys/cddl/contrib/opensolaris/uts/common/fs/zfs/vdev_label.c:410:14, shift exponent 64 is too large for 64-bit type 'unsigned long long'

Coverage

KCOV – Coverage Sanitizer

- Coverage sanitizer
- Inserts function calls to trace:
  - The start of basic blocks
  - On comparison operations
- Comparison tracing includes values being compared
  - Useful for finding what input data to try changing
- Committed to:
  - OpenBSD on 19 August 2018
  - FreeBSD on 12 January 2019
  - NetBSD on 23 February 2019

KCOV – PC Tracing

Count	PC	PC	PC	…
348	0xffffffff81595cf0	0xffffffff8155c2f0	0xffffffff8155ccb0	…

- Starts with the number of entries
- Each entry contains an address in the basic block
  - Probably the return address of the inserted function
- Each field is:
  - uintptr_t on OpenBSD
  - uint64_t on FreeBSD and NetBSD (as kcov_int_t)

KCOV – Comparison Tracing

Count	Type	Arg	Arg	PC	Type	Arg	Arg	PC	…
348	0x2	0x10	0x20	0xffffffff…	0x7	0x8080	0x8080	0xffffffff…	…

- Starts with the number of entries
- Each entry contains:
  - A comparison type – encodes width and if comparing with a const
  - Two arguments
  - An address near the comparison
    - Probably the return address of the inserted function
- Each field is:
  - uintptr_t on OpenBSD
  - uint64_t on FreeBSD and NetBSD (as kcov_int_t)

KCOV – User interface

1. User opens /dev/kcov
2. Sets the buffer size with an ioctl
3. mmaps the buffer
4. Enables tracing within the thread being traces
   - May not be the same thread (or process) as opened the device
5. Zeros the first entry in the buffer
6. Runs the traced operations
7. Disables tracing
8. May repeat from 4
9. Unmaps the buffer
10. Closes the device

Address Space

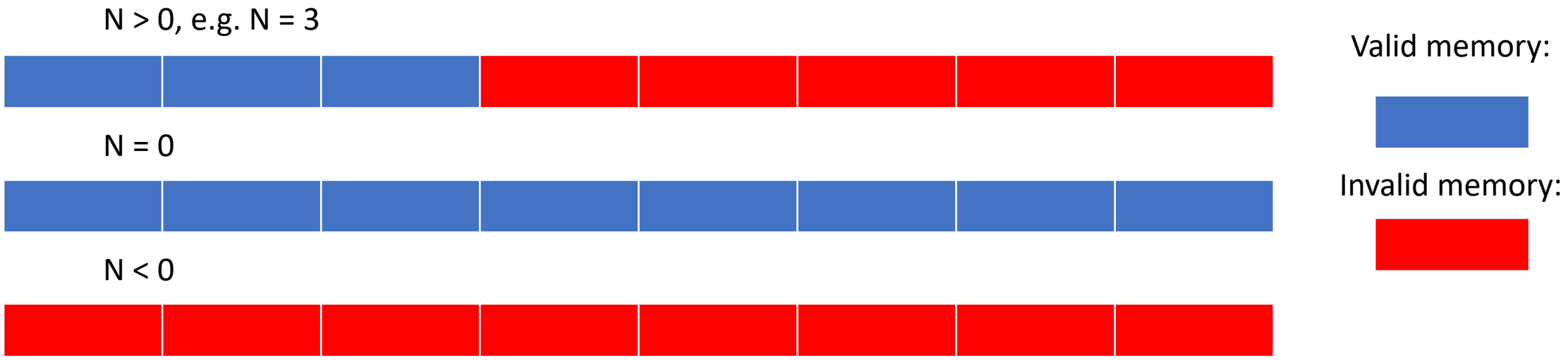
KASAN – Address Space Sanitizer

- Checks memory accesses are in bounds
- Uses a shadow map to mark valid memory
  - 1 shadow byte for each 8 bytes
- Can mark the first 1-8 bytes as valid
  - Must be contiguous
- Can mark all bytes as invalid
  - Includes data on why, e.g. stack padding

KASAN – Address Space Sanitizer

1 byte in the shadow map (signed)

N



KASAN – Address Space Sanitizer

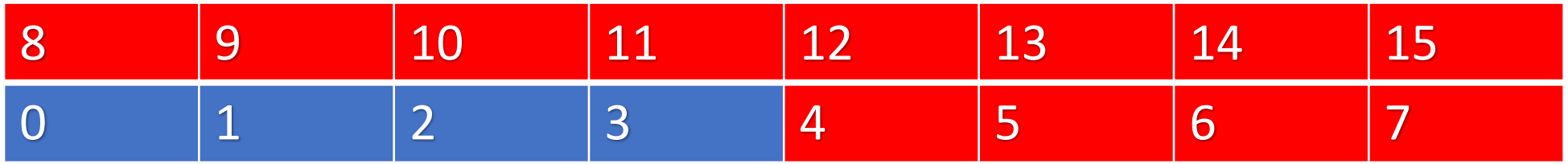
- All allocations are now aligned to at least an 8 byte boundary
- Allocations are rounded up to an 8 byte boundary
  - Memory past the end of the allocation is marked as unusable
- One of more 8 byte blocks of unusable memory after the allocation

KASAN – Example

```
void get_data(int *output, size_t count);

int example(void) {
        int ret, *data = malloc(sizeof(int), M_TEMP, M_WAITOK);

        get_data(data, 1);
        ret = *data;

        free(data, M_TEMP);
        return (ret);
}
```

KASAN – Example

Allocation ↓

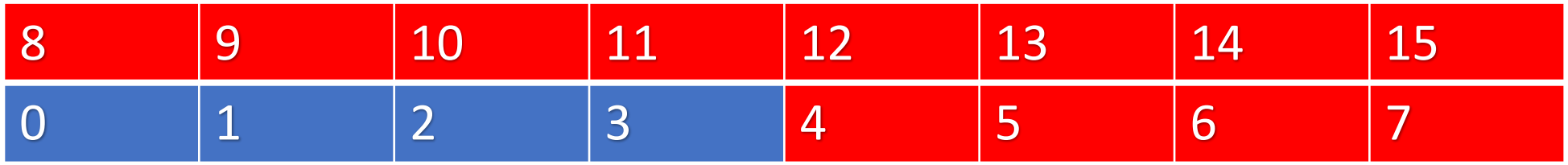


- Allocated 4 bytes
- Aligned data to an 8 byte boundary
- Padded to 8 bytes
- Allocated 8 bytes of invalid data

KASAN – Example

Allocation ↓

Shadow map →



…
0xFF
0x04
…

- Allocated up to 8 bytes
- Padded up to 8 bytes
- 8 bytes of padding after the allocation

KASAN – Example

Allocation ↓

8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

Shadow map →

…
0xFF
0x04
…

- A load or store that includes bytes 4-15 will be detected
  - Can warn with printf or panic
- A load or store past byte 15 may or may not be detected
  - It depends on if it has been allocated

KASAN – Example

```
void get_data(int *output, size_t count);

int example(void) {
        int ret, *data = malloc(sizeof(int), M_TEMP, M_WAITOK);

        get_data(data, 1);
        ret = data[1];              /* Out of bounds: access past the end of data */

        free(data, M_TEMP);
        return (ret);
}
```

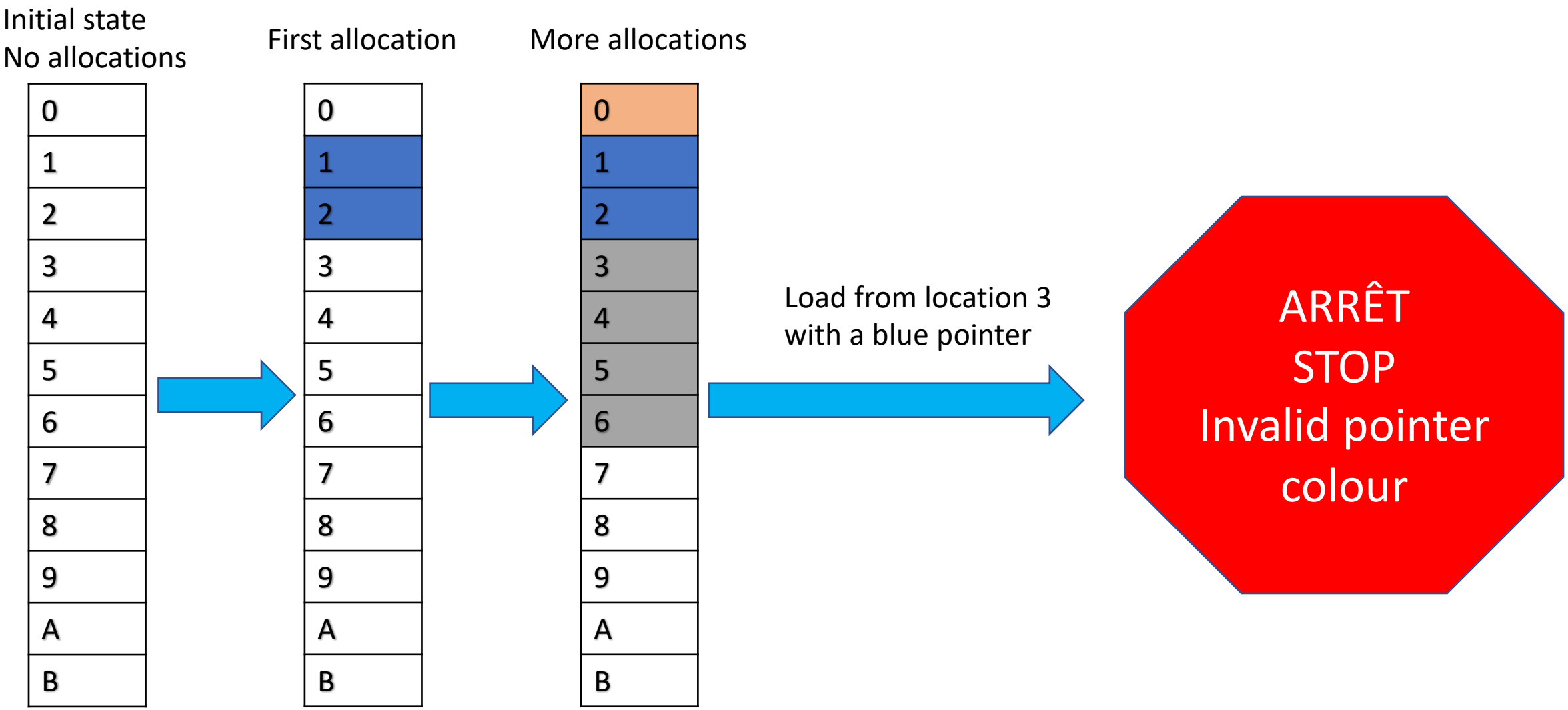
KASAN

- Committed to NetBSD 20 August 2018
- FreeBSD has a Google Summer of Code student working on it

KHWASAN – Hardware Assisted ASAN

- An arm64 specific extension
- Enable Top Byte Ignore in the kernel
  - The top 8 bits of a pointer are ignored
- Store an 8 bit tag in the top byte
  - One tag is reserved for free memory
- Store the same tag in the shadow map
- Allocate random tags when allocating memory

KHWASAN – Example



KHWASAN – Hardware Assisted ASAN

- The shadow map is 1 byte per 16 allocated bytes
- Allocations are 16 byte aligned and sized
- Not able to detect slightly out of bounds access
  - Within a 16 byte block
- Can probabilistically detect larger out of bounds access of allocated memory
  - Just under 1/256 probability of an incorrect tag match

CHERI

- Add bounds and permissions to pointers to create capabilities
- Capabilities are non-forgeable
- Can only reduce bounds and permissions
- Capabilities can only be derived from other capabilities
  - At least one will be provided to the kernel on boot
- With CheriABI all pointers are capabilities
  - See Brooks' talk tomorrow for more information

CHERI

- CHERI with CheriABI can detect all out-of-bounds access
  - Not just slightly out of bounds like KASAN
  - No tag collision like KHWASAN
- Research on narrowing bounds more
  - Stopping buffer overflows for variables within a struct

Memory

KMSAN – Memory Sanitizer

- Checks use of uninitialized values in the kernel
- Use is defined as:
  - Conditionals
  - Pointer dereference
  - Copied to userspace

KMSAN – Memory Sanitizer

```
int a;
int b = a; /* Not a use */

copyout(&b, u, sizeof(b)); /* An uninit use */
```

KMSAN – Memory Sanitizer

```
int a;


if (flag)
        a = 1;
c = a + b;      /* Not a use of a */
if (flag)
        copyout(&c, u, sizeof(c)); /* A use, don't report */
```

KMSAN – Memory Sanitizer

```
struct config *example(void)
{
        struct config *conf;

        conf = malloc(sizeof(*conf), M_TEMP, M_WAITOK);
        init_config(conf);
}
void init_conf(struct config *conf)
{
        if (conf->flag != 0)          /* An uninit use */
                do_something();
}
```

KMSAN – Memory Sanitizer

- As with KASAN it uses a shadow map
- 1 bit per byte
  - Set when poisoned (uninitialized)
- Memory is poisoned by default
- malloc(9) will unpoison memory with M_ZERO
- Writing a constant unpoisons memory
- Shadow state is propagated

KLEAK – NetBSD memory leak detector

- Similar in concept to KMSAN
  - Find copying uninitialized memory to userspace
- Uses in-band signalling
  - Use a magic value, then check for it when copying to userspace
- Uses the coverage sanitizer instrumentalization to poison the stack
- Prone to false positives
  - In-band value may be legitimately in the data
  - Cleaver choice of value to reduce this chance

Threading

KTSAN – Thread Sanitizer

- Find data races
- Still a work in progress by Google in Linux
  - May be on hold

Why add Kernel Sanitizers?

Why add Kernel Sanitizers?

- Find and fix more bugs
- KCOV allows kernel fuzzing
- Improves fuzzing by making bugs easier to find

Syzkaller

- A system call fuzzer from Google
- Supports many kernels including FreeBSD, NetBSD, and OpenBSD
- Finds new ways to panic the kernel from userspace
- Google hosts an instance on their infrastructure

syzbot

FreeBSD

fixed bugs (16)

Instances:

Name	Active	Uptime	Corpus	Coverage	Crashes	Execs	Kernel build			syzkaller build		
							Commit	Freshness	Status	Commit	Freshness	Status
ci-freebsd-main	now	5h46m	8936	81982	137	1563723	85eaade9	13h46m		2755003a	9d02h	

upstream (47):

Title	Repro	Bisected	Count	Last	Reported
panic: inp_leave_group: imf_sources not empty (2)	syz		2	2h31m	3h01m
Fatal trap 12: page fault in uma_dbg_free			1	1d05h	1d05h
panic: Duplicate free of ADDR from zone ADDR(16) slab ADDR(241)			1	1d16h	1d16h
Fatal trap 12: page fault in __mtx_assert (2)			1	1d18h	1d18h
Fatal trap 9: general protection fault in sys_nlm_syscall			2	2d03h	4d22h
Fatal trap 12: page fault in fifo_close			17	17h43m	5d15h
panic: Assertion lock == sq->sq_lock failed at /syzkaller/managers/m…			1	6d03h	6d03h
panic: Memory modified after free ADDR(256) val=0 @ ADDR	syz		4	1d05h	6d11h
Fatal trap 12: page fault in uma_dbg_alloc			1	6d12h	6d12h
panic: mtx_lock() of destroyed mutex at sys/kern/sys_socket.c:LINE	syz		1	6d19h	6d19h
panic: mtx_lock() of spin mutex (null) @ /syzkaller/managers/main/k…			1	6d19h	6d19h
Fatal trap 12: page fault in link_elf_search_symbol			1	7d00h	7d00h
Fatal trap 12: page fault in ip_output			2	6d14h	7d13h
panic: Most recently used by temp			4	3d12h	7d23h
panic: Bad tailq NEXT(ADDR->tqh_last) != NULL			1	8d10h	8d10h
panic: Most recently used by tty	syz		11	4d00h	9d00h
panic: Most recently used by ip6opt			1	11d	11d

syzbot

Linux

fixed bugs (1240)

Instances:

Name	Active	Uptime	Corpus	Coverage	Crashes	Execs	Kernel build			syzkaller build		
							Commit	Freshness	Status	Commit	Freshness	Status
ci-upstream-bpf-kasan-gce	now	1h53m	10023	284396	163	2141078	d72386fe	1d08h		95dfd515	2h21m	
ci-upstream-bpf-next-kasan-gce	now	1h53m	10406	287727	249	2742186	35c99ffa	2d15h		95dfd515	2h21m	
ci-upstream-gce-leak	now	24m	11460	330887	11	87189	a6a4b66b	10h06m		95dfd515	2h21m	
ci-upstream-kasan-gce	now	39m	72945	4602079	466	912856	a6a4b66b	10h06m		95dfd515	2h21m	
ci-upstream-kasan-gce-386	now	57m	22792	430747	14	1914041	a6a4b66b	10h06m		95dfd515	2h21m	
ci-upstream-kasan-gce-root	now	14m	74600	4827346	440	892084	a6a4b66b	10h06m		95dfd515	2h21m	
ci-upstream-kasan-gce-selinux-root	now	30m	73966	4668186	282	1229467	a6a4b66b	10h06m		95dfd515	2h21m	
ci-upstream-kasan-gce-smack-root	now	48m	72906	5032570	236	1588729	a6a4b66b	10h06m		95dfd515	2h21m	
ci-upstream-kmsan-gce	now	1h07m	56237	3125847	614	465976	2b51a114	4h01m		95dfd515	2h21m	
ci-upstream-linux-next-kasan-gce-root	now	1h53m	72275	5189609	156	1956910	b1d6682e	8h14m		95dfd515	2h21m	
ci-upstream-net-kasan-gce	now	1h53m	19059	429673	49	4444259	35c99ffa	2d15h		95dfd515	2h21m	
ci-upstream-net-this-kasan-gce	now	1h53m	17762	409496	36	3111905	510e2ced	15h46m		95dfd515	2h21m	
ci2-upstream-usb	now	5h28m	770	32489	466	747197	43151d6c	580d		2755003a	9d02h	

open (505):

Title	Repro	Bisected	Count	Last	Reported
general protection fault in ext4_mb_initialize_context			3	18d	1d01h
KASAN: use-after-free Write in xfrm_hash_rebuild			1	7d05h	1d01h
KASAN: use-after-free Write in __xfrm_policy_unlink (2)			2	4d09h	1d01h
WARNING: locking bug in inet_autobind			1	1d11h	1d06h
WARNING: locking bug in udpv6_pre_connect			1	4d21h	1d06h

Syzkaller

- Will combine system calls to try finding new paths through the kernel
- Understands arguments
  - E.g. read takes a file description, a pointer, and a length
- Will try to mix syscalls in interesting ways
  - Pass a socket into something that doesn’t take a socket
- Very good at panicking the kernel
- Will try to find a reproducer
- Adding a sanitizer makes it easier to find memory issues

panic: ffs_blkfree_cg: freeing free block
Status: fixed on 2019/04/29 23:55
Reported-by: syzbot+36fd786cb3ab88f18c9b@syzkaller.appspotmail.com
Fix commit: a7a455c2 Optimize lseek(SEEK_DATA) on UFS.
First crash: 60d, last: 60d

similar bugs (1):

Kernel	Title	Repro	Bisected	Count	Last	Reported	Patched	Status
freebsd	panic: ffs_blkfree_cg: freeing free block (2)			1	17d	17d	0/1	upstream: reported on 2019/04/30 12:16

Sample crash report:

```
panic: ffs_blkfree_cg: freeing free block
cpuid = 0
time = 1552872502
KDB: stack backtrace:
db_trace_self_wrapper() at db_trace_self_wrapper+0x47/frame 0xfffffe0020dfc150
vpanic() at vpanic+0x1e0/frame 0xfffffe0020dfc1b0
panic() at panic+0x43/frame 0xfffffe0020dfc210
ffs_blkfree_cg() at ffs_blkfree_cg+0x6e9/frame 0xfffffe0020dfc2d0
ffs_blkfree() at ffs_blkfree+0x15e/frame 0xfffffe0020dfc350
ffs_indirtrunc() at ffs_indirtrunc+0x724/frame 0xfffffe0020dfc450
ffs_indirtrunc() at ffs_indirtrunc+0x856/frame 0xfffffe0020dfc530
ffs_truncate() at ffs_truncate+0x17c3/frame 0xfffffe0020dfc720
ufs_setattr() at ufs_setattr+0x918/frame 0xfffffe0020dfc7c0
VOP_SETATTR_APV() at VOP_SETATTR_APV+0xc2/frame 0xfffffe0020dfc7f0
vn_truncate() at vn_truncate+0x23f/frame 0xfffffe0020dfc930
kern_ftruncate() at kern_ftruncate+0x13b/frame 0xfffffe0020dfc980
amd64_syscall() at amd64_syscall+0x436/frame 0xfffffe0020dfcab0
fast_syscall_common() at fast_syscall_common+0x101/frame 0xfffffe0020dfcab0
--- syscall (0, FreeBSD ELF64, nosys), rip = 0x42132a, rsp = 0x7fffffffea88, rbp = 0x2 ---
KDB: enter: panic
[ thread pid 762 tid 100093 ]
Stopped at      kdb_enter+0x6a: movq    $0,kdb_why
```

All crashes (5):

Manager	Time	Kernel	Commit	Syzkaller	Config	Log	Report	Syz repro	C repro
ci-freebsd-main	2019/03/18 01:31	freebsd	8b17fbc2	f8757044		log	report	syz	C
ci-freebsd-main	2019/03/18 01:14	freebsd	8b17fbc2	f8757044		log	report		
ci-freebsd-main	2019/03/18 05:00	freebsd	8b17fbc2	f8757044		log	report		

Syzkaller

- Emails a per-project mailing list with new issues
- Fixes should be tagged in the commit
- Will check the issue is fixed

- Join the appropriate list if you care about kernel quality

AFL – American Fuzzy Lop

- A file format fuzzer
  - Can change a file and see if any new paths are found
- Test patches for KCOV to support AFL
- Before starting clear the buffer
- On each basic block:
  - Calculate (hash(old_ptr) ^ hash(new_ptr)) % buffer_length
  - Increment this entry
- Patched AFL to talk to kcov

AFL – Fuzzing UFS

- Tried fuzzing a 128K UFS image
- Just mount and unmount the image
- Very slow
  - ~60 mounts/second
  - Around 12 days to try all single bitflips

Conclusion

Conclusion – Sanitizers

- FreeBSD, NetBSD, and OpenBSD have KCOV and KUBSAN
- NetBSD has KASAN, with it planned for FreeBSD
- Other sanitizers need work
- Will make bugs easier to find

Conclusion – Fuzzing

- Google runs a syscall fuzzer on FreeBSD, NetBSD, and OpenBSD
- Look through the reports & fix the code
- AFL may be useful in the future, but currently is too slow

Questions?