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 \dots/sys/vm/uma_core.c:1746:8, member access within misaligned address \texttt{0xfffff8087ffde7c0} for type 'struct uma_zone' which requires 128 byte alignment

• NULL pointer dereference:
  • UBSan: Undefined Behavior in \dots/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 \dots/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

<table>
<thead>
<tr>
<th>Count</th>
<th>PC</th>
<th>PC</th>
<th>PC</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>348</td>
<td>0xffffffff8159cf0</td>
<td>0xffffffff8155c2f0</td>
<td>0xffffffff8155ccb0</td>
<td>...</td>
</tr>
</tbody>
</table>

- 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

<table>
<thead>
<tr>
<th>Count</th>
<th>Type</th>
<th>Arg</th>
<th>Arg</th>
<th>PC</th>
<th>Type</th>
<th>Arg</th>
<th>Arg</th>
<th>PC</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>348</td>
<td>0x2</td>
<td>0x10</td>
<td>0x20</td>
<td>0xffffffff...</td>
<td>0x7</td>
<td>0x8080</td>
<td>0x8080</td>
<td>0xffffffff...</td>
<td>...</td>
</tr>
</tbody>
</table>

- 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. mmap 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

N > 0, e.g. N = 3

Valid memory:

N = 0

Invalid memory:

N < 0
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

```c
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 ↓

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Allocation ↓</th>
<th>Shadow map →</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
KASAN – Example

• 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

```c
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

Initial state
No allocations

First allocation

More allocations

Load from location 3 with a blue pointer

ARRÊT
STOP
Invalid pointer colour
KH WASAN – 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-forgable
• 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

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

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

```c
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
### fixed bugs (16)

#### Instances:

<table>
<thead>
<tr>
<th>Name</th>
<th>Active</th>
<th>Uptime</th>
<th>Corpus</th>
<th>Coverage</th>
<th>Crashes</th>
<th>Execs</th>
<th>Commit</th>
<th>Kernel build</th>
<th>syzkaller build</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ci-freebsd-main</td>
<td>now</td>
<td>5h46m</td>
<td>8936</td>
<td>81982</td>
<td>137</td>
<td>1563723</td>
<td>85eaade9</td>
<td>13h46m</td>
<td>2755003a</td>
<td>9d02h</td>
</tr>
</tbody>
</table>

#### upstream (47):

<table>
<thead>
<tr>
<th>Title</th>
<th>Repro</th>
<th>Bisected</th>
<th>Count</th>
<th>Last</th>
<th>Reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>panic: inp _leave_group: imf_sources not empty</td>
<td>syz</td>
<td></td>
<td>2</td>
<td>2h31m</td>
<td>3h01m</td>
</tr>
<tr>
<td>Fatal trap 12: page fault in uma DBG_free</td>
<td></td>
<td></td>
<td>1</td>
<td>1d05h</td>
<td>1d05h</td>
</tr>
<tr>
<td>panic: Duplicate free of ADDR from zone ADDR(16) slab ADDR(241)</td>
<td></td>
<td></td>
<td>1</td>
<td>1d16h</td>
<td>1d16h</td>
</tr>
<tr>
<td>Fatal trap 12: page fault in __mtx_assert (2)</td>
<td></td>
<td></td>
<td>1</td>
<td>1d18h</td>
<td>1d18h</td>
</tr>
<tr>
<td>Fatal trap 9: general protection fault in sys_nlm_syscall</td>
<td></td>
<td></td>
<td>2</td>
<td>2d03h</td>
<td>4d22h</td>
</tr>
<tr>
<td>Fatal trap 12: page fault in fifo_close</td>
<td></td>
<td></td>
<td>17</td>
<td>17413m</td>
<td>5d15h</td>
</tr>
<tr>
<td>panic: Assertion lock == sq-&gt;sq_lock failed at syzkaller/managers/m…</td>
<td></td>
<td></td>
<td>1</td>
<td>6d03h</td>
<td>6d03h</td>
</tr>
<tr>
<td>panic: Memory modified after free ADDR(256) val=0 @ ADDR</td>
<td>syz</td>
<td></td>
<td>4</td>
<td>1d05h</td>
<td>6d11h</td>
</tr>
<tr>
<td>Fatal trap 12: page fault in uma DBG_alloc</td>
<td></td>
<td></td>
<td>1</td>
<td>6d12h</td>
<td>6d12h</td>
</tr>
<tr>
<td>panic: mtx_lock() of destroyed mutex at sys/kern/sys_socket.c:LINE</td>
<td>syz</td>
<td></td>
<td>1</td>
<td>6d19h</td>
<td>6d19h</td>
</tr>
<tr>
<td>panic: mtx_lock() of spin mutex (null) @ syzkaller/managers/main/k…</td>
<td></td>
<td></td>
<td>1</td>
<td>6d19h</td>
<td>6d19h</td>
</tr>
<tr>
<td>Fatal trap 12: page fault in link_elf_search_symbol</td>
<td></td>
<td></td>
<td>1</td>
<td>7d00h</td>
<td>7d00h</td>
</tr>
<tr>
<td>Fatal trap 12: page fault in ip_output</td>
<td></td>
<td></td>
<td>2</td>
<td>6d14h</td>
<td>7d13h</td>
</tr>
<tr>
<td>panic: Most recently used by temp</td>
<td></td>
<td></td>
<td>4</td>
<td>3d12h</td>
<td>7d23h</td>
</tr>
<tr>
<td>panic: Bad tailq NEXT(ADDR-&gt;tqh_last) != NULL</td>
<td></td>
<td></td>
<td>1</td>
<td>8d10h</td>
<td>8d10h</td>
</tr>
<tr>
<td>panic: Most recently used by tty</td>
<td>syz</td>
<td></td>
<td>11</td>
<td>4d00h</td>
<td>9d00h</td>
</tr>
<tr>
<td>panic: Most recently used by ip6opt</td>
<td></td>
<td></td>
<td>1</td>
<td>11d</td>
<td>11d</td>
</tr>
</tbody>
</table>
### fixed bugs (1240)

#### Instances:

<table>
<thead>
<tr>
<th>Name</th>
<th>Active</th>
<th>Uptime</th>
<th>Corpus</th>
<th>Coverage</th>
<th>Crashes</th>
<th>Execs</th>
<th>Kernel build</th>
<th>syzkaller build</th>
</tr>
</thead>
<tbody>
<tr>
<td>ci-upstream-bpf-kasan-gce</td>
<td>now</td>
<td>1h53m</td>
<td>10023</td>
<td>284396</td>
<td>163</td>
<td>2141078</td>
<td>d72386fe</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-bpf-next-kasan-gce</td>
<td>now</td>
<td>1h53m</td>
<td>10406</td>
<td>287727</td>
<td>249</td>
<td>2742186</td>
<td>35c99ffa</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-gce-leak</td>
<td>now</td>
<td>24m</td>
<td>11460</td>
<td>330887</td>
<td>11</td>
<td>87189</td>
<td>a6a4b666</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-kasan-gce</td>
<td>now</td>
<td>39m</td>
<td>72945</td>
<td>4602079</td>
<td>466</td>
<td>912856</td>
<td>a6a4b666</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-kasan-gce-386</td>
<td>now</td>
<td>57m</td>
<td>22792</td>
<td>430747</td>
<td>14</td>
<td>1914041</td>
<td>a6a4b666</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-kasan-gce-root</td>
<td>now</td>
<td>14m</td>
<td>74600</td>
<td>4827346</td>
<td>440</td>
<td>892084</td>
<td>a6a4b666</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-kasan-gce-selinux-root</td>
<td>now</td>
<td>30m</td>
<td>73966</td>
<td>466186</td>
<td>282</td>
<td>1229467</td>
<td>a6a4b666</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-kasan-gce-smack-root</td>
<td>now</td>
<td>48m</td>
<td>72906</td>
<td>5032570</td>
<td>236</td>
<td>1588729</td>
<td>a6a4b666</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-kmsan-gce</td>
<td>now</td>
<td>1h07m</td>
<td>56237</td>
<td>3125847</td>
<td>614</td>
<td>465976</td>
<td>2b51a114</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-linux-next-kasan-gce-root</td>
<td>now</td>
<td>1h53m</td>
<td>72275</td>
<td>5189609</td>
<td>156</td>
<td>1956910</td>
<td>b1d6682e</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-net-kasan-gce</td>
<td>now</td>
<td>1h53m</td>
<td>19059</td>
<td>429673</td>
<td>49</td>
<td>4444259</td>
<td>35c99ffa</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci-upstream-net-this-kasan-gce</td>
<td>now</td>
<td>1h53m</td>
<td>17762</td>
<td>409496</td>
<td>36</td>
<td>3119905</td>
<td>5102ced</td>
<td>95dfd515</td>
</tr>
<tr>
<td>ci2-upstream-usb</td>
<td>now</td>
<td>5h28m</td>
<td>770</td>
<td>32489</td>
<td>466</td>
<td>747197</td>
<td>43151d6c</td>
<td>27550034a</td>
</tr>
</tbody>
</table>

#### open (505):

<table>
<thead>
<tr>
<th>Title</th>
<th>Repro</th>
<th>Bisected</th>
<th>Count</th>
<th>Last</th>
<th>Reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>general protection fault in ext4_mbr_initialize_context</td>
<td>3</td>
<td>18d</td>
<td>1d01h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Write in xfrm_hash_rebuild</td>
<td>1</td>
<td>7d05h</td>
<td>1d01h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KASAN: use-after-free Write in __xfm_policy_unlink (2)</td>
<td>2</td>
<td>4d09h</td>
<td>1d01h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARNING: locking bug in inet Autobind</td>
<td>1</td>
<td>1d11h</td>
<td>1d06h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARNING: locking bug in udpv6_pre_connect</td>
<td>1</td>
<td>4d21h</td>
<td>1d06h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Title</th>
<th>Repro</th>
<th>Bisection</th>
<th>Count</th>
<th>Last</th>
<th>Reported</th>
<th>Patched</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>freebsd</td>
<td>panic: ffs_blkfree_cg: freeing free block</td>
<td></td>
<td></td>
<td>1</td>
<td>17d</td>
<td>17d</td>
<td>0/1</td>
<td>upstream: reported 2019/04/30 12:16</td>
</tr>
</tbody>
</table>

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 0xffffffff0020dfc150
    vpanic() at vpanic+0x1e0/frame 0xffffffff0020dfc1b0
    panic() at panic+0x43/frame 0xffffffff0020dfc210
    ffs_blkfree_cg() at ffs_blkfree_cg+0x6e9/frame 0xffffffff0020dfc2d0
    ffs_blkfree() at ffs_blkfree+0x15e/frame 0xffffffff0020dfc350
    ffs_indirtrunc() at ffs_indirtrunc+0x724/frame 0xffffffff0020dfc450
    ffs_indirtrunc() at ffs_indirtrunc+0x856/frame 0xffffffff0020dfc530
    ffs_truncate() at ffs_truncate+0x17c3/frame 0xffffffff0020dfc720
    ufs_setattr() at ufs_setattr+0x818/frame 0xffffffff0020dfc7c0
    VOP_SETATTR_APV() at VOP_SETATTR_APV+0xc2/frame 0xffffffff0020dfc7f0
    vn_truncate() at vn_truncate+0x23f/frame 0xffffffff0020dfc930
    kern_ptruncate() at kern_ptruncate+0x13b/frame 0xffffffff0020dfc980
    amd64_syscall() at amd64_syscall+0x436/frame 0xffffffff0020dfcb0
    fast_syscall_common() at fast_syscall_common+0x101/frame 0xffffffff0020dfeab0
    --- syscall (0, FreeBSD ELF64, nosys), rip = 0x42132a, rsp = 0x7fffffffef88, rbp = 0x2 ---
KDB: enter: panic
[ thread pid 762 tid 100093 ]
Stopped at kern_ptruncate+0x6a: movq $0,kdb_why
```

All crashes (5):

<table>
<thead>
<tr>
<th>Manager</th>
<th>Time</th>
<th>Kernel</th>
<th>Commit</th>
<th>Syzkiller</th>
<th>Config</th>
<th>Log</th>
<th>Report</th>
<th>Syz repro</th>
<th>C repro</th>
</tr>
</thead>
<tbody>
<tr>
<td>ci-freebsd-main</td>
<td>2019/03/18 01:31</td>
<td>freebsd</td>
<td>8b17f6c2</td>
<td>88757044</td>
<td></td>
<td></td>
<td></td>
<td>syz</td>
<td>C</td>
</tr>
<tr>
<td>ci-freebsd-main</td>
<td>2019/03/18 01:14</td>
<td>freebsd</td>
<td>8b17f6c2</td>
<td>88757044</td>
<td></td>
<td></td>
<td></td>
<td>syz</td>
<td>C</td>
</tr>
<tr>
<td>ci-freebsd-main</td>
<td>2019/03/18 05:00</td>
<td>freebsd</td>
<td>8b17f6c2</td>
<td>88757044</td>
<td></td>
<td></td>
<td></td>
<td>syz</td>
<td>C</td>
</tr>
</tbody>
</table>
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 \((\text{hash(old_ptr) } \text{\textasciicircum} \text{ hash(new_ptr)}) \% \text{buffer_length})\)
  • 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?