
Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benfamin Chelf
SOSP 2001

Presented by Zachary Bischof

NORTHWESTERN UNIVERSITY
Introduction

- Bugs are a problem
- Difficult to identify in systems code
  - Rules are unclear
  - Correctness is unknown
- Methods for identifying bugs:
  - Type systems
  - Specifications
  - High-level compilation
  - Dynamic invariant inference
Deviant Behavior

- If correctness rules are known, we can check them with an extended compiler
  - Manually finding rules is difficult
  - Want to extract it automatically, but how?

- Find incorrect behavior without knowing correct behavior
  - Cross check statements in code
  - Identify contradiction
  - Common behavior is probably correct behavior (hopefully)
Beliefs

- Automatically generate beliefs
  - Extract beliefs from the source code
  - Compare beliefs in different sections
  - Contradictions in beliefs
    - May be an error
    - May be a coincidence
    - May also identify sections of programmer confusion

- Two types of beliefs:
  - MUST beliefs
  - MAY beliefs
MUST beliefs
- Directly implied by code
- Check using internal consistency
- Contradiction of MUST beliefs directly implies an error
- Examples:
  - $x = a / b$
    - $b$ is non-zero
  - $*ptr$
    - $ptr$ is not null
  - unlock(lck)
    - $lck$ has been acquired
Beliefs

- **MAY beliefs**
  - Observed features, suggested by code
  - May be a coincidence, treat as MUST beliefs
  - E.g. ordering
    - ‘a();’ followed by ‘b();’ MAY mean a() and b() must be paired
    - Enclosure in locks may mean locking is required
  - Lock followed by use of a and b, b may be a coincidence
  - Separate coincidences from valid beliefs using probability
May Beliefs (cont’d)
- Use statistical analysis to filter out coincidences
  \[ z(n, e) = \frac{e/n - p_0}{\sqrt{p_0 \times (1 - p_0)/n}} \]
- Measures the amount of deviation in beliefs
- Error cases have some number of counter-examples
- Also useful to rank \( z(n, n - e) \)
  - Inversion shows beliefs that are almost never true
  - Such beliefs may also be errors
- Stop when the number of false pos is too high
Three possible beliefs for a pointer
- Null, not-null, or unknown

Checker rules
- A dereference adds not-null to set of beliefs
  - Error if the previous belief set was null
- A comparison check implies two things
  - Before the comparison the belief is unknown
  - After the comparison (ptr == null), belief is null in true branch and non-null in false branch
Null Pointers

- **Check-then-use (79 errors 26 false pos)**
  
  ```c
  /* 2.4.1: drivers/isdn/avmb1/capidrv.c */
  1: if (card == NULL) {
  2:     printk(KERN_ERR "capidrv-%d: ... %d!\n",
  3:             card->contrnr, id);
  4: }
  ```

- **Use-then-check (102 bugs, 4 false)**

  ```c
  /* 2.4.7: drivers/char/mxser.c */
  struct mxser_struct *info = tty->driver_data;
  unsigned flags;
  if (!tty || !info->xmit_buf)
      return 0;
  ```
Spreading Beliefs, Lock Inference

1: lock l;  // Lock
2: int a, b;  // Variables potentially
    // protected by l

3: void foo() {
4:     lock(l);  // Enter critical section
5:     a = a + b;  // MAY: a,b protected by l
6:     unlock(l);  // Exit critical section
7:     b = b + 1;  // MUST: b not protected by l
8: }

9: void bar() {
10:    lock(l);
11:    a = a + 1;  // MAY: a protected by l
12:    unlock(l);
13: }

14: void baz() {
15:    a = a + 1;  // MAY: a protected by l
16:    unlock(l);
17:    b = b - 1;  // MUST: b not protected by l
18:    a = a / 5;  // MUST: a not protected by l
19: }

EECS 443 - Advanced Operating Systems
Redundancy

- Contradiction/redundant checks (24 bugs, 10 false)
  
  ```c
  /* 2.4.7/drivers/video/tdfxfb.c */
  fb_info.regbase_virt = ioremap_nocache(...);
  if(!fb_info.regbase_virt)
    return -ENXIO;
  fb_info.bufbase_virt = ioremap_nocache(...);
  /* [META: meant fb_info.bufbase_virt!] */
  if(!fb_info.regbase_virt) {
    iounmap(fb_info.regbase_virt);
  }
  ```

- Assume code should be useful
  - Useless statements identify areas of confusion

  ```c
  /* 2.4.5-ac8/net/appletalk/aarp.c */
  da.s_node = sa.s_node;
  da.s_net = da.s_net;
  ```
Kernel pointers are safe, user pointers are not
  – Any violation is a security hole
  – How to find user pointers?
    💡 Use a similar analysis to finding null pointers

*ptr implies a non-null pointer
  – copyin(ptr)/copyout(ptr) suggests a user pointer
  – Belief is propagated throughout code

Found 24 security bugs in Linux, 18 in OpenBSD
/* drivers/net/appletalk/ipddp.c:ipddp_ioctl */
case SIOCADDIPDDPRT:
    return ipddp_create(rt);
case SIOCDELIPDDPRT:
    return ipddp_delete(rt);
case SIOFCINDIPDDPRT:
    if(copy_to_user(rt, ipddp_find_route(rt),
                   sizeof(struct ipddp_route)))
        return -EFAULT;

- rt is treated as a user pointer, but is dereferenced before it is checked
- Area of confusion for programmer
- 1:1 ratio of false positives
Kernel code must check for failure

- Assumptions for checker:
  - Assume all functions can fail
  - If the result of a function is ignored or used without checks, “error”
  - If the result of a function is checked before use, “checked”
- A high ratio of check to error messages implies checking is necessary
/ * ipc/shm.c:map_zero_setup */
  if (IS_ERR(shp = seg_alloc(...)))
    return PTR_ERR(shp);

/* 2.4.0-test9:ipc/shm.c:newseg */
   NOTE: checking 'seg_alloc' */
if (!(shp = seg_alloc(...)))
  return -ENOMEM;
int ipc_addid(..., struct kern_ipc_perm* new)
  new->cuid = new->uid = current->euid;
  new->gid = new->cgid = current->egid;
  ids->entries[id].p = new;
Use-after-free errors can cause heavy damage
- Want to keep track of “free” calls
- Must identify undocumented free functions
  - Assume all functions contain free
    foo(p); foo(p); foo(p); bar(p); bar(p); bar(p);
    *p = x; *p = x; *p = x; p = null; p = null; *p = x;
- foo has fewer deviations than bar, bar has higher rank for error detection
- Error may be the caused by an unexpected return path
- Found 23 free errors, 11 false pos
Deallocation Errors

- Returning a freed pointer

```c
/* fs/proc/generic.c:proc_symlink */
ent->data = kmalloc(...);
if (!ent->data) {
    kfree(ent);
    goto out;
}
out:
return ent;
```
/ * drivers/block/cciss.c:cciss_ioctl */
if (iocommand.Direction == XFER_WRITE) {
    if (copy_to_user(...)) {
        cmd_free(NULL, c);
        if (buff != NULL) kfree(buff);
        return( -EFAULT);
    }
}
if (iocommand.Direction == XFER_READ) {
    if (copy_to_user(...)) {
        cmd_free(NULL, c);
        kfree(buff);
    }
}
cmd_free(NULL, c);
if (buff != NULL) kfree(buff);
Finding a-b Pairs

- a(); ... b(); implies a MAY belief that a() must always be followed by b()

- Assume all a-b sequences are valid
  - Note: use latent specifications and prefiltering to restrict to likely pairs

- Scan for all function calls
  - “check” for each a() ... b() sequence
  - “error” for all lone a() calls

- Rank errors

- Found 23 errors and 11 false positives
Finding a-b Pairs

```
drivers/sound/trident.c:trident_release:
lock_kernel();
card = state->card;
dmabuf = &state->dmabuf;
VALIDATE_STATE(state);
```

- Kernel lock not always released on some error paths within VALIDATE_STATE(state);
Finding a-b Pairs

```c
/* drivers/sound/esssolo1.c:solo1_midi_release */
static int solo1_midi_release(...) {

    ...
    lock_kernel();
    if (file->f_mode & FMODE_WRITE) {
        add_wait_queue(&s->midi.owait, &wait);
        for (;;) {
            __set_current_state(TASK_INTERRUPTIBLE);
            spin_lock_irqsave(&s->lock, flags);
            count = s->midi.ocnt;
            spin_unlock_irqrestore(&s->lock, flags);
        ...
        if (file->f_flags & O_NONBLOCK) {
            remove_wait_queue(...);
            set_current_state(TASK_RUNNING);
            /* did not release lock! */
            return -EBUSY;
        }
    }

    ...
    unlock_kernel();
    return 0;

    ■ Possible to return without releasing Kernel lock
```
Summary

- Extract code beliefs, find errors without knowing the truth
  - MUST belief contradictions are errors
  - MAY beliefs should be treated as MUST beliefs and then ranked by their confidence rating
- Flag areas with redundancy/useless code
  - High chance of error
    - Could be a typo
    - Programmer confusion could mean errors are nearby