Linaro Connect
MADRID 2024 | MAY 14-17 2024

Static Analysis and You!

(Smatch mostly)

The kernel is a heavy user of static analysis

- 2-4% of patches come from static analysis
- A similar percent of static analysis patches were backported to stable kernels since 2016
- Saves developer time
- Prevents issues for customers

Tools

- GCC / Clang W=1
- Checkpatch
- Coverity
- Cpp Check
- Sparse
- Coccinelle
- Smatch

Sparse

- Good for tagging data and flagging misuse of data

- Endian data
- User space pointers
- IO Mem pointers

- Smatch uses it as a C front end

Coccinelle

- It’s easy to write Coccinelle checks
- Generates patches for you
- Useful for kernel hardening
- Nicer than Smatch for checking macros

Smatch

- Flow analysis
- Cross functions analysis
- Works on pre-processed code

Flow analysis

Flow analysis is the math to understand code.

```
if (x == 1)
        y = 2;
else
        y = 3;

if (x == 2)
        __smatch_implied(y);  // ← prints “y = 3”
```

What Smatch tracks

- Value ranges: x = 0,20-40
- What values can be controlled by the user
- Variable comparisons: x < y
- Buffer sizes: p is 40-50 bytes
- If a variable has been capped to an unknown value
- If we are in an impossible code path
- If a function will return -EINVAL if we pass -100
- This function returns negative error codes

Cross Function Analysis

```
{ "__request_region", ALLOC,   1, "$", &valid_ptr_min_sval, &valid_ptr_max_sval },
{ "release_resource", RELEASE, 0, "$->start" },
{ "__release_region", RELEASE, 1, "$" },
```

Cross Function Analysis

- smdb.py function
- smdb.py return_states function
- smdb.py functions struct_name member
- smdb.py where struct_name member

Difficult problems

- Recursion
- It takes a too much memory to track how every variable is related to every other variable

- Smatch is slow
- Smatch takes shortcuts parsing loops
- Smatch doesn’t understand threads
- Smatch is bad at tracking data in arrays

Unsolvable Problems

- Bug vs Feature
- Specification issues
- Firmware issues
- Hardware issues

False positives

- After you fix the bugs, you are left with 100% false positives
- Only review new warnings
- Don’t silence false positives (unless it makes the code more readable)

False positives

- Any warning which is not a bug is a false positive:

```
unsigned int x;
...
if (x < 0)
        return -EINVAL;
...
if (x < 0 || x > 9)
        return -EINVAL;
```

False positives

- Any warning which is not a bug is a false positive:

```
int i;
...
for (i = 0; i < ARRAY_SIZE(foo); i++) {
```

Solvable problem #1 = vs ==

- Bad: if (x = NULL) {
- Good: if (x == NULL) {

Solvable problem #1 = vs ==

- Yoda Code

```
if (NULL == x) {
```

NO!

Solvable problem #1 = vs ==

- Testing

Solvable problem #1 = vs ==

- GCC: Add parentheses to show it is intentional

```
while ((x = frob()) {
```

Solvable problem #1 = vs ==

- Side note about intentionality:

```
	int ret = 0;
	...
	if (x == 3)
		goto done;
	...
done:
	return ret;
```

Solvable problem #1 = vs ==

- Side note about intentionality:

```
        int ret;
        ...
        if (x == 3) {
                ret = 0;
                goto done;
        }
        ...
done:
        return ret;
```

Solvable problem #1 = vs ==

- More side notes about intentionality part 2:

Bad:

```
	if (!ret)
		return ret;
```

Good:

```
	if (!ret)
		return 0;
```

Solvable problem #1 = vs ==

- Back to the talk:

- GCC: Add parentheses to show it is intentional

```
while ((x = frob()) {
```

Solvable problem #1 = vs ==

- Checkpatch: Move assignments out of if statements
- Checkpatch: Write NULL checks as if (!x) {

```
x = frob();
if (!x) {
```

Solvable problem #1 = vs ==

- Smatch: Complain about if (x = CONSTANT) {
- Smatch: Complain about if (x = &foo) {

```
-		result = ASSERT(offset = sizeof(buffer),
+		result = ASSERT(offset == sizeof(buffer),
```

Solvable problem #1 = vs ==

RESULTS

- 27 bugs total since 2005
- Most bugs caught by static analysis

Further ideas:

- if (x == a || y = b || z == c) {
- ASSERT(x = 1);
- Double parentheses for ternary operations
- = vs == in parameters: frob(x = 1);
- Reversed the other way, using == when = is intended

Solvable problem #2 tun.c

Famous Bug: CVE-2009-1897

```
struct sock *sk = tun->sk;

if (!tun)
        return POLLERR;
```

- -fno-delete-null-pointer-checks
- mmap_min_addr changes
- Smatch and Coccinelle checks for inconsistent NULL checking

Solvable problem #3 goto fail

Famous Bug: CVE-2014-1266

```
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail; // ← OOPS COPY AND PASTE
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

Solvable problem #3 goto fail

- GCC/Smatch: missing curly braces
- Smatch: unreachable code
- Smatch: inconsistent indenting

Related checks

- Smatch: missing error code

Failed approach

- Looking for duplicate lines

Typical Smatch Check

- Add a hook for allocations
- Add a hook for frees
- Add a hook for returns statements
  - Is this an error path?
  - Are there any variables still on allocated state?

- Write it as quickly and broadly as possible
- Rewrite it to filter out false positives

Reviewing CVEs

- Most CVEs are race conditions
- Add a new function to the list of functions that get user data
- A bug is a bug

```
commit 6d97e55f7172303082850c1de085d06fc1e57d17
Author: Dan Carpenter <error27@gmail.com>
Date:   Mon Oct 11 19:24:19 2010 +0200

    vhost: fix return code for log_access_ok()
```

size_add() size_mul()

- Using the results for math
- Saving the results in anything besides unsigned long
- Passing the results to a function that takes an unsigned int

Scoped based cleanup

```
struct gpio_sim_device *dev __free(kfree) = kzalloc(sizeof(*dev), GFP_KERNEL);
```

- Not initializing the pointer to NULL (checkpatch?)
- Re-assigning uncleaned up pointers
- Declaring a variable as function scope when it is assigned in a loop
- Adding an automatic cleaned up pointer to a list

Variables i and j that aren’t incremented

- Match declarations
  - Is this variable named “i” or “j”
- Match when “i” or “j” are modified
  - If we assign 0 to the variable mark it as an &set
  - If we set it to anything else mark it as &okay
- If we have variable which is &set but never &okay then print a warning

Suspicious negatives

```
    case AXI_DAC_PHASE_TONE_1:
    case AXI_DAC_PHASE_TONE_2:
        return axi_dac_phase_set(st, chan, buf, len,
-                                private - AXI_DAC_PHASE_TONE_2);
+                                private == AXI_DAC_PHASE_TONE_2);
```

Takeaways

- Once a month review fixes and brainstorm about how they could have been detected faster
- Nibble away at the bugs
- If it’s stupid but it works then it isn’t stupid

Linaro Connect
MADRID 2024 | MAY 12-17 2024

Thank you

For more information
Dan Carpenter <dan.carpenter@linaro.org>
Mailing list: <smatch@vger.kernel.org>