Making C less dangerous

Keeping current, 9/12/2018

Presented by Raphael Finkel

Based on

- https://lwn.net/SubscriberLink/763641/c9a04da2a33af0a3/
- https://developers.slashdot.org/story/18/09/01/2311248/how-linuxs-kernel-developers-make-c-less-dangerous
- https://www.hpe.com/us/en/insights/articles/making-c-less-dangerous-1808.html
- https://raphlinus.github.io/programming/rust/2018/08/17/undefined-behavior.html

Strict aliasing

- Aliasing problems: overlaying a `struct` with a buffer of, say, `int`.

```
struct msg_t {
    int a, b;
} msg;
char *buffer = &msg;
```

- What's can go wrong?
  - Optimization assumes that two pointers of two different types can't overlap.
  - The compiler might place `msg.a` in a register in a situation where it is used heavily, such as in a loop.
  - Access to `buffer[0]` might not use that register.
- Fix the problem by using a union

```
union {
    struct msg_t msg;
    char asBuffer[sizeof(msg_t)/sizeof(char)]
}
```

- Also worry about *endianness* when overlaying with `char`.

Undefined behavior in C

- Uninitialized local variables (garbage based on previous memory use)
  - *gcc* fix: `-finit-local-vars`
- `void` pointers to callable typed functions (a type-system weakness)
- Poorly designed library routines (`memcpy()` doesn't take/update a "destination-remaining size").

Dynamic-length arrays

```
int size = 8192
char buf[size];
buf[bad] = foo;
```

- Can overwrite return address or other stack frame; can circumvent guard pages and stack canaries
- Actually slower (13%) than static-length arrays.
- *gcc* warning: `-Wvla`

Fall-through of `switch` cases

- A common error (67 such bugs reported in Linux kernel): forgetting `break` in a `switch` branch.
  - Other languages inherit the same poor design (Java, JavaScript).
  - But some automatically insert `break`, require `fallthrough` if desired (Go)
- *gcc* warning: `-Wimplicit-fallthrough`; mark intentional fall-through with a comment.
- unrelated error with `switch`: unreachable initialization statements (before the first `case`.
  - *gcc* warning: `-Wswitch-unreachable`

Arithmetic overflow

- Signed integer arithmetic overflow is usually an error.
  - *Unsigned* overflow is often intentional, as in hash-function computation
- The hardware does not generate a trap (unlike divide-by-zero error).
- But code can check by investigating condition codes.
  - *gcc* fix: `-fsanitize=signed-integer-overflow`
  - Fast: undetectable time cost
  - Big: warnings increase size by 6%
- Use macros to explicitly check individual operations

```
if (check_add_overflow(a, b, &c)) return -EOVERFLOW;
```

Bounds checking

- C array indices are not checked for bounds
  - It is easy to commit an off-by-one error.
  - Even negative indices are valid, but are usually erroneous.
- The kernel has some explicit checks
  - `copy_to_user()`: less than 1% speed cost
  - `strncpy()`: about 2% speed cost
    - Don't use for null-terminated strings; the final null may not fit.
    - null-pads entire destination, even if not needed.
    - `strlcpy()` always places at least one null at the end.
- Hardware memory tagging is much faster; available on SPARC.

Control-flow integrity

- Forward-edge vulnerability: Methods stored in the heap are often called without checking the function prototype.
- Backward-edge vulnerability: An attack might overwrite the return address.
  - Shadow stack holds a copy of the return address
  - Function prologue copies the return address to the shadow stack
  - Function epilogue compares shadow return address with ordinary return address
  - *Clang* fix: `-fsanitize=shadow-call-stack`
    - Hardware support for signed return address in ARM v8.3a
    - *gcc*: `-msign-return-address`

The Linux Kernel Self-Protection Project (KSPP)

- Purpose: protect the *kernel* (not *userspace*) from attack.
- The Linux kernel is mostly written C, because it generates fast code. Architecture-dependent parts (memory management, interrupt handling, …) are written in assembler.
- Status
  - Nearly eradicated variable-length arrays.
  - Steady progress on marking fall-through on `switch` branches.
  - Waiting for compiler help on always-initialized local variables.
  - Explicit arithmetic overflow detection for memory allocation.
  - Waiting for hardware support for bounds checking.
  - Forward-edge control-flow integrity: in progress; works on Android.
  - Backward-edge control-flow integrity: shadow stack on Android (ARM); waiting for hardware support for other platforms.

Variants of C

- Semi-portable C
  - C is “a portable assembly language”
  - heavy use of `#ifdef` and *autoconf*.
  - type punning is OK so long as sizes align.
- Standard (ANSI) C: A compromise
  - many enhancements to provide type security
  - must still pay attention to correct use of pointers
    - avoid memory errors: use-after-free, double-free, out-of-bounds access
  - introduces “undefined behavior”
    - shift-past-bitwidth: “x << 64 is allowed to crash, subtly corrupt memory, or connect to a server to transfer money out of your account.”
    - signed integer overflow, reading uninitialized memory, computing (not just dereferencing) an out-of-bounds pointer, type punning through pointers, `https://blog.regehr.org/archives/213`

Other pitfalls due to bad language design

- The = and == operators look similar, and wherever one is valid, so is the other.
- Curly braces are not required on `if` branches, or `for` or `while` bodies. (Go requires braces)
  - A maintainer must add braces to enlarge the branch or body.
  - The programmer could accidentally place `;` before the body.
- Function prototypes are not required.
- The auto-increment and auto-decrement operators are confusing

```
j = j++ // what does this mean?
```

- Conflating pointers and arrays
  - If you pass an array to a function, the function treats it as a pointer, and bounds checking is not possible.
- Strings require a null terminator.

```
myString = (char *)malloc(stringLength+1);
```