FORTIFY_SOURCE=3

A standalone portable header-based implementation

FORTIFY_SOURCE=3

Memory-safety-ish the boomer way for glibc smugs.

FORTIFY_SOURCE=3

A Band-Aid for old embedded codebases.

FORTIFY_SOURCE=3

Come for the security, stay for the revolting C hacks.

FORTIFY_SOURCE=?

Main idea: Using the compiler's knowledge to enforce invariants, mostly via `__builtin_object_size`.

- `FORTIFY_SOURCE=1` added in 2004 by Jakub Jelinek from Red Hat
  - requires (gcc ≥ 4.0 or clang ≥) and glibc ≥ 2.3.4
  - checks at compilation-time only
- `FORTIFY_SOURCE=2` "some conforming programs might fail."
  - checks at compilation-time and runtime
- `FORTIFY_SOURCE=3` added in glibc 2.33 in 2021
  - takes advantage of `__builtin_dynamic_object_size`
  - requires (gcc ≥ 12.0 or clang ≥ 9.0) and glibc ≥ 2.33

Giving credit where credit is due

- *sin* from 2f30 created fortify-header in 2015
- I took over around 2023:
  - added `FORTIFY_SOURCE=3` support
  - added modern functions attributes
  - added a comprehensive testsuite
  - fixed some minor bugs
  - added more functions
  - added clang support, based on q66's work

Implementation details

Quizz time: how would **you** do it?

- headers only
- “portable” code
- C doesn’t have functions overloading

Functions “overloading” in C, yay!

1. ~~Horrible~~ creative macros:

```
#define _FORTIFY_STR(s) #s
#define _FORTIFY_ORIG(p, fn) __typeof__(fn) \
    __orig_##fn __asm__(_FORTIFY_STR(p) #fn)
#define _FORTIFY_FN(fn) _FORTIFY_ORIG(__USER_LABEL_PREFIX__, fn)
#define _FORTIFY_FN(fn) _FORTIFY_FNB(fn); _FORTIFY_INLINE
```

2. ~~Horrible~~ magical compiler extensions:

```
#include_next
```

3. Used like this:

```
_FORTIFY_FN(memcpy) void* memcpy(void* dest, const void* src, size_t n)
```

More ~~ugly hacks~~ advanced engineering

- Labels are in assembly, but they're portable, since they're, well, … labels.
  - They also prevent weird symbol names mangling issues
- On clang, the pass_object_size/pass_dynamic_object_size attributes are used to pass down arguments size
- Ask the compiler as much as we can to inline our wrappers:

```
#define _FORTIFY_INLINE static __inline__ \

        __attribute__((__always_inline__,__artificial__,__overloadable__))
```

- Let's see an example together: `memcpy`

Annotations

```
__fh_access(write_only, 1, 3)

__fh_access(read_only, 2, 3)

#if __has_builtin(__builtin_memcpy)

__diagnose_as_builtin(__builtin_memcpy, 1, 2, 3)

#endif
```

Prototype and compile-time checks

```
_FORTIFY_FN(memcpy) void* memcpy(void* _FORTIFY_POS0 __od,
                                 const void* _FORTIFY_POS0 __os,
                                 size_t __n)
__error_if((__fh_bos(__od, 0) < __n), "'memcpy' called with `n`
bigger than the size of `d`.")
{
```

Use of _chk builtins if available

```
#if __has_builtin(__builtin___memcpy_chk) && USE_NATIVE_CHK

        return __builtin___memcpy_chk(__od, __os, __n,
__fh_bos(__od, 0));

#else

[…]

#endif

}
```

Actual implementation

```
__fh_size_t __bd = __fh_bos(__od, 0);
__fh_size_t __bs = __fh_bos(__os, 0);
char *__d = (char *)__od;
const char *__s = (const char *)__os;

if __fh_overlap(__d, __bd, __s, __n)
        __builtin_trap();
if (__n > __bd || __n > __bs)
        __builtin_trap();
return __builtin_memcpy(__od, __os, __n);
```

Example of test

```c
#include "common.h"
#include <string.h>

int main(int argc, char** argv) {
  char buffer[8] = {0};
  memcpy(buffer, "1234567890", sizeof(buffer) - 1);
  puts(buffer);
  CHK_FAIL_START
  memcpy(buffer, "1234567890", argc);
  CHK_FAIL_END
  puts(buffer);
  return ret;
}
```

And now what?

- Writing code is easy, writing bug-less code less so
  - Kudos to Chimera Linux and Gentoo Hardened for being ~~guinea pigs~~ early adopters
- Harden more functions and catch more bugs types
  - See the open issues
- Add more tests
  - various architectures
  - various compilers and various versions
- Maybe some benchmarks
  - Most checks are happening at build-time, and are O(1), but pretty numbers are pretty.
- ~~Harass~~ nudge upstreams to update!

Bonus petty dunks on glibc

- Some functions from the `printf` family are parsing `/proc/self/maps` to see if the format specifier is in read-only memory.
- The stack canary is stored below the stack.
- `argv` is also stored below the stack + `*** stack smashing detected ***: ./my_binary terminated`
  - known since at least 2010
- Inline metadata in its allocator, enjoy the gazillion house-of-talk-at-blackhat techniques
- More ~~dissing~~ unbiased data here

Sources

- Source of truth
- Github mirror used for continuous integration and issues tracking
- Blogpost on the topic