Towards Secure Input Handling in C/C++ Code

RIPE 71, Bucharest

Jan Včelák • jan.vcelak@nic.cz • 2015 November 19

CZ DOMAIN REGISTRY

Overview

- **Defensive programming**
- Static analysis
  - Coverity
  - Cppcheck
  - clang-analyzer
- **Dynamic analysis**
  - Valgrind
  - Sanitizers in clang
- **Coverage-guided fuzzy testing**

Defensive programming

- Code:
  - Arbitrary input sanitization
  - Buffer boundary checking
  - Error codes checking
- Development process:
  - Writing well-aranged and readable code
  - Doing code reviews
  - Writing tests

Defensive programming – Example

```
wire_ctx_t rr = wire_ctx_init_const(rr_data, rr_size);

uint16_t rtype  = wire_ctx_read_u16(&rr);
uint16_t rclass = wire_ctx_read_u16(&rr);
uint32_t ttl    = wire_ctx_read_u32(&rr);
uint32_t rdlen  = wire_ctx_read_u32(&rr);

if (rr.error) {
     return KNOT_EMALF;
}

/* further procesing */
```

Dynamic analysis

- Instruments running code
- Faulty state has to be reached
- Usually single-purposed analysis (memory, threads, etc.)
- Available analyzers:
  - Valgrind

    Memcheck, Cachegrind, Callgrind, Massif, Hellgrind, DRD
  - clang sanitizers

    Address Sanitizer, Memory Sanitizer, Leak Sanitizer,
    Thread Sanitizer, Dataflow Sanitizer, Control Flow Integrity

Dynamic analysis – Memory

Valgrind: Memcheck

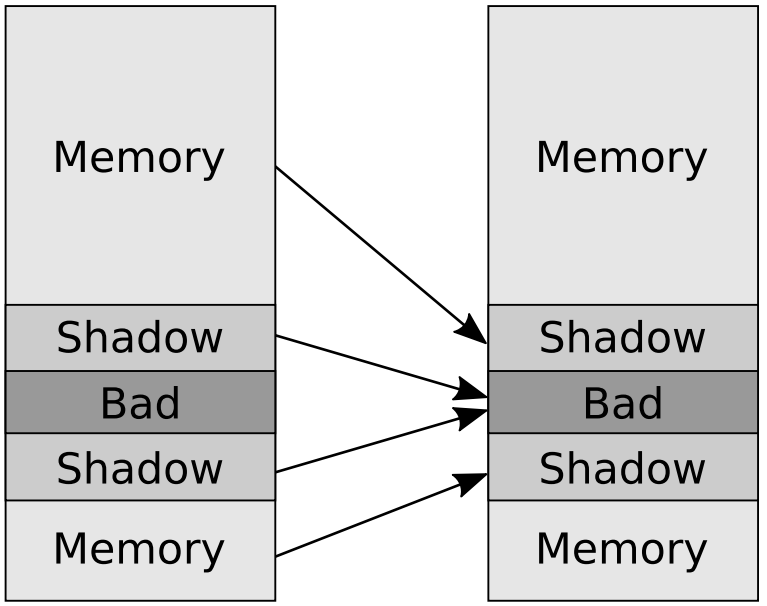
- Virtual machine, compiled code translation
- Out-of-boundary access, uninitialized reads, memory leaks
- Limitted support for global objects and stack variables (SCGCheck)
- Costs: 10–50× run time, 12–18× used memory

clang: Address Sanitizer

- Instrumentation code added during compilation, *-fsanitize=address*
- Checks possible for global objects and stack variables
- No uninitialised reads checking (Memory Sanitizer)
- Costs: 1.5–3× run time, 2–4× used memory

Address Sanitizer – Essentials

- ASAN reserves all usable memory
- Any memory access is checked
- Keeps track of allocations in the *Shadow* part of the memory (one-eight of all memory)



- Example for 8-byte structure access:

```
shadow = (addr << 3) + offset;
if (*shadow) {
    report_and_crash(addr);
}
```

Coverage-guided fuzzy testing

- Useful where arbitrary input is processed
- More effective than blind fuzzy testing
- Simplified execution loop:
  1. Take an input from the corpus
  2. Perform some modification on the input
  3. Run the code with the new input
  4. Take a look at the coverage
  5. Update the corpus if the input produced an interesting code path

Coverage-guided fuzzy testing – Tools

American Fuzzy Lop (AFL)

- Uses LLVM/clang hooks
- Single-shot and persistent fuzzing
- Synthesizing complex input semnantics
- Experimental testing of black-box binaries (in QEMU)

LibFuzzer

- Heavily inspired by AFL
- Depends on coverage support in clang sanitizers
- Real coverage, no approximation
- Instruction traces (experimental)

LibFuzzer – Like a Pro in Three Slides (1)

- Write the test driver:

```
#include <stddef.h>
#include <stdint.h>
#include "libknot/pkt.h"

int LLVMFuzzerTestOneInput(const uint8_t *data,
                           size_t size)
{
    knot_pkt_t *pkt = knot_pkt_new(data, size, NULL);
    knot_pkt_parse(pkt, 0);
    knot_pkt_free(&pkt);

    return 0;
}
```

LibFuzzer – Like a Pro in Three Slides (2)

- Compile LibFuzzer:

```
% cd ~devel/llvm-3.7.0/lib
% clang -g -O2 -c -std=c++11 -IFuzzer Fuzzer/*.cpp
% ar rcs Fuzzer.a Fuzzer*.o
```

- Compile the test driver:

```
% clang -fsanitize=address -fsanitize-coverage=edge \
        -g -O2 -o fuzz fuzz.c Fuzzer.a -lstdc++
```

- Start fuzzing:

```
% ./fuzz
```

LibFuzzer – Like a Pro in Three Slides (3)

- Find a bug:

```
==30094==ERROR: AddressSanitizer: heap-buffer-overflow on address ...
WRITE of size 1 at 0x60600000efbf thread T0
    #0 0x4d3843 in LLVMFuzzerTestOneInput /tmp/fuzz.c:12:26
    ...
    #9 0x41d648 in _start (/tmp/fuzz+0x41d648)
...
==30094==ABORTING
DEATH:
0x61,0x62,0x2a,0x83,0x61,0x73,
ab*\x83as
CRASHED; file written to crash-1cef67f42be69e7d3f01c3d5e1f5112ce3e66ece
Base64: YWIqg2Fz
```

- Fix the bug:

```
% ./fix-bug-automagically.php || vim
```

- Rinse and Repeat.

Thank you!

Jan Včelák • jan.vcelak@nic.cz • www.knot-dns.cz