Targeted Memory Runtime Analysis

David Piché

December 8th, 2022

Polytechnique Montreal

DORSAL Laboratory

Agenda

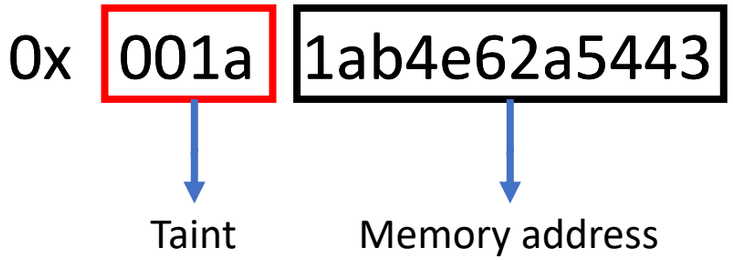
1. Introduction
2. Previous work
3. Our general approach
    1. Using Ptrace
    2. Using Ptrace and shared virtual memory
    3. Using instruction emulation
4. Results
5. Future Works

Introduction

- Memory issues in C/C++ are still prevalent
  - Use-after-free
  - Memory leaks
  - Out-of-bound writes
  - And much more…

Previous Work

- X86_64 architecture
- Minimal approach to recreate datawatch:
  - Overwrite the *malloc/realloc* to add a taint.
  - Tainted pointers: use bits 47 to 63 for pointer tainting.



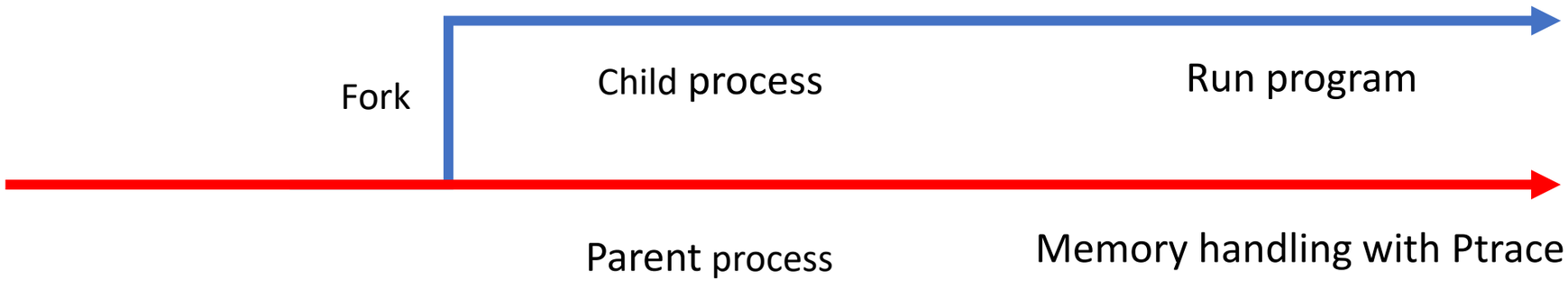
Our general approach: Memory Handler

- For each memory access, we need to:
  - Use a signal handler (SIGSEGV, SIGBUS…)
  - Identify the right register with the tainted address
    - Disassemble using capstone
  - Do bounds checking
  - Un-taint the address
  - Execute the faulty instruction
  - Re-taint the address

- How can we do those last 2 steps?

Our general approach: Ptrace

- Use Ptrace with 2 different processes
  - The child process runs the program with the special allocators
  - The parent process takes care of memory handling
  - Ptrace used for communication between processes and single-step



Our general approach: Ptrace

- 2 processes do not share the same virtual memory!
- We need to do multiple Ptrace PEEKDATA calls with each tainted memory access
- Additional overhead!

Our general approach: Ptrace and shared virtual memory

- Using clone() instead of fork(), we can use the same virtual memory for the two processes (CLONE_VM)

- Less overhead, as the Ptrace PEEKDATA calls are no longer needed

- Ongoing development

Our general approach: Instruction emulation

- Using Olivier's Libpatch, we can directly emulate the instructions
- Reduces significantly the need for signal handling
- Ongoing development

Result: Overhead

- For each tainted memory access, the approach with Ptrace needs ~100µs.
- With the Ptrace with clone() approach, we can remove multiple Ptrace PEEKDATA calls per memory access (each being ~2µs).
- With libpatch, performance improved, as we only have to call the special handler once for each tainted address.

Future Works

- Use the upper 16 bits to store useful information:
  - Object id: identify objects more prone to memory errors
  - Use tags instead of object ids

- **Targeted** memory analysis
  - Taint some memory allocations based on parameters (size?)