Prologue: Evaluation

Please participate and help to improve the lab

https://befragung.zqa.tu-dresden.de/uz/de/sl/T7f6W2wfEFp7
Some Ethymology/History

Rear Admiral Grace Murray Hopper

1947: "First actual case of bug being found"
Some Ethymology/History

Rear Admiral Grace Murray Hopper

1947: “First actual case of bug being found”
**Definitions**

**Bug** ... flaw in a computer system that results in unexpected behavior

**Debugging** ... process of searching and fixing deviations from the expected behavior
Debugging is not only finding living creatures in an electronic device:

• Program crash
• Wrong result
• Slow execution
How to debug?
How to debug? Example: Digging
How to debug? Example: Digging
How to debug? Example: Digging
How to debug? Example: Digging
How to debug? Example: Digging
Debugging Tools

- strace
- ltrace
- gdb
- valgrind
- perf
- ptrace
- and even more...
Tracing System Calls — \texttt{strace}

Inspect system calls performed by a program

- **Filtering:** \texttt{strace -e}
- **Timing:** \texttt{strace -t[tt] / strace -T}
- **Statistics:** \texttt{strace -c}
1. Which system calls are performed when you run /bin/ls?
2. How many calls are performed?
3. Why so many?
Tracing library calls — ltrace

Inspect all calls to *shared* libraries

- Filtering: `ltrace -e`
- Timing: `ltrace -t[tt] / ltrace -T`
- Statistics: `ltrace -c`
$ wget https://os.inf.tu-dresden.de/Studium/SysProg/SS2023/debugging/strace.tar.xz

$ tar -xJf strace.tar.xz

$ cd strace

Make it print “SUCCESS”!

Hints: file, strace, ltrace, objdump
Problem: Memory Leaks

1. Allocate memory buffer
2. Use the buffer
3. Stop using the buffer
4. (Optional) Loose pointer to the buffer
5. Rinse and repeat
Dynamic Linker

- Recall static linking vs. dynamic linking

Details: man ld.so
Dynamic Linker

- Recall static linking vs. dynamic linking
- Resolves symbols by searching for libraries in LD_LIBRARY_PATH

Details: man ld.so
Dynamic Linker

• Recall static linking vs. dynamic linking
• Resolves symbols by searching for libraries in LD_LIBRARY_PATH
• LD_PRELOAD
  • Force loading of libraries
  • Loaded before any other *dynamic* library
  • Application has no choice

Details: man ld.so
Detecting Memory Leaks

• Use LD_PRELOAD to let the leaky program call custom implementations of malloc/free
• Track malloc/free information to report memory leaks at program termination
• Use the real malloc/free to perform the actual work
Interfacing with the Dynamic Linker

```c
void* dlopen(const char* filename, int flag);
char* dlerror(void);
void* dlsym(void* handle, const char* symbol);
int dlclose(void* handle);
```

And link with `libdl`, i.e. `gcc ... -ldl`
C/C++ Function Pointers

void* (*real_malloc)(size_t) = NULL;
C/C++ Function Pointers

void* (*real_malloc)(size_t) = NULL;

• Function return type
C/C++ Function Pointers

```c
void* (*real_malloc)(size_t) = NULL;
```

- **Function return type**
- **Variable name**
```c
void* (*real_malloc)(size_t) = NULL;
```

- Function return type
- Variable name
- Function parameter types
C/C++ Function Pointers

```c
void* (*real_malloc)(size_t)= NULL;

typedef void* (*malloc_ptr)(size_t);
malloc_ptr real_malloc = NULL;
```

- Function return type
- Variable name
- Function parameter types
Finding the Real malloc

#define _GNU_SOURCE
#include <dlfcn.h>

// Inside the wrapper function
{
    static malloc_ptr real_malloc = NULL;
    real_malloc = (malloc_ptr) dlsym(RTLD_NEXT, "malloc");
}
Assignment №3

  → tar.xz
- In the malloc/free wrappers in mallocWrap.c:
  - Track memory management information
  - Redirect work to the real malloc and free;
- Upon exit, print all pointers (and sizes) that were not free’d;
- You will need to be notified upon exit():
  - Wrap it
  - Or use atexit()

Hint: Be careful about using malloc/free yourself (indirectly).
An anecdote

1. Bug report on strange sound on mp3 flash website
An anecdote

1. Bug report on strange sound on mp3 flash website
2. Located in libflashplayer.so
An anecdote

1. Bug report on strange sound on mp3 flash website
2. Located in libflashplayer.so
3. Reason: Use of memcpy for overlapping regions
An anecdote

1. Bug report on strange sound on mp3 flash website
2. Located in libflashplayer.so
3. Reason: Use of memcpy for overlapping regions
4. Should use memmove, but plugin is closed source
Linus’ Workaround

http://bugzilla.redhat.com/show_bug.cgi?id=638477#c38

1. Write your own memcpy similar to memmove
2. gcc -O2 -c mymemcpy.c
3. ld -G mymemcpy.o -o mymemcpy.so
4. LD_PRELOAD mymemcpy.so /opt/google/chrome/google-chrome &
Valgrind

Binary recompilation framework (Valgrind core) with various tools:

**MemCheck** memory checks (default)

**CacheGrind** cache profiling

**CallGrind** call graph analysis

**Helgrind** race condition detection

How do you pronounce “Valgrind”? (from FAQ)

The “Val” as in the word “value”. The “grind” is pronounced with a short “i” — ie. “grinned” (rhymes with “tinned”) rather than “grined” (rhymes with “find”). Don’t feel bad: almost everyone gets it wrong at first.
Valgrind

Binary recompilation framework (Valgrind core) with various tools:

**MemCheck** memory checks (default)

**CacheGrind** cache profiling

**CallGrind** call graph analysis

**Helgrind** race condition detection

---

How do you pronounce “Valgrind”?  

(from FAQ)

The “Val” as in the word “value”. The “grind” is pronounced with a short “i” — ie. “grinned” (rhymes with “tinned”) rather than “grined” (rhymes with “find”). Don’t feel bad: almost everyone gets it wrong at first.
Assignment №4

Analyze some programs with Valgrind:

- Get https://os.inf.tu-dresden.de/Studium/SysProg/SS2023/debugging/
  → valgrind.tar.xz
- Use build.sh
Static Checker


→ xz

scan-build

1. Install the Clang static analyser (e.g. `apt install clang-tools-<version>`) 
2. Run `scan-build make` to analyze code 
3. Run `scan-view` to see the report 

Lists of static analyzers

- https://spinroot.com/static/
Compiler Sanitizers

Additional libraries which are able to detect: race conditions, memory bugs, undefined behavior, ...

Assignment №5: Address Sanitizer

1. Install libasan (e.g. apt install libasan<version>)
2. Run make asan (re-builds all programs with -fsanitize=address)
3. Run the programs

Details: man gcc/-fsanitize
The GNU Debugger

Interactive debugger (gdb):

• Breakpoints, Watchpoints
• Single-stepping, Reverse-stepping
• Inspect/modify registers & memory
• Scripting

Best with binaries containing debug info, e.g. compiled with the -g (or, better, -ggdb3) option
Basics

- **r[un] [args] [>...][<...]**
- **start [args] [>...][<...]**
- **starti [args] [>...][<...]**
- **q[uit]**
- **h[elp] [command]**
Breakpoints & Watchpoints

- `b[reak] {function | line | *address} [if condition]`
- `wa[tch] {variable | *address}`
- `info {b[reak] | wa[tch]}`
- `commands {id(s)}`
- `c[ontinue]`
Inspecting the Program

- `list` [+|−][N] — show program code
- `disassemble` — disassemble current function
- `info registers` — show register content
- `print [/FMT] {variable | expression}` — evaluate and print variable or expression
- `x/FMT {address}` — examine memory
- `bt` — backtrace
Going Forward

- `step` — step to next source line
- `stepi` — step to next assembler instruction
- `next` — step to next source line, proceeding through function calls
- `nexti` — step to next assembler instruction, proceeding through function calls
- `finish` — run to return from current function
Going Backwards

- `record full` — start full execution recording
- `record stop` — stop execution recording
- `rs[tep]` — step to previous source line
- `rs[tep]i` — step to previous assembler instruction
- `rn[ext]` — step to previous line, proceeding through function calls
- `rn[ext]i` — step to previous assembler instruction, proceeding through function calls

See also: https://rr-project.org/
Remote Debugging

• GDB can connect to remote GDB servers
  • Via TCP or serial line
  • `set target remote {address:port}`

• Heavily used in OS/embedded development

• Qemu, Bochs/x86, Valgrind, etc. contain their own GDB servers
Alternate UI

- [tui] layout {asm | src | regs}
- https://github.com/cyrus-and/gdb-dashboard
- https://sourceware.org/gdb/wiki/GDB Front Ends
Scripting

- Run `gdb -ex {gdb_command}`
- Write GDB commands into a text file & run `gdb -x {file}`
- `define mycommand`
- Python API
Assignment №6

https://os.inf.tu-dresden.de/Studium/SysProg/SS2023/debugging/gdb.tar.xz

There are 4 versions of the Sieve of Eratosthenes

But only one works properly

What’s wrong with the rest?