CS033 Project Gear-Up: Traps

The Project

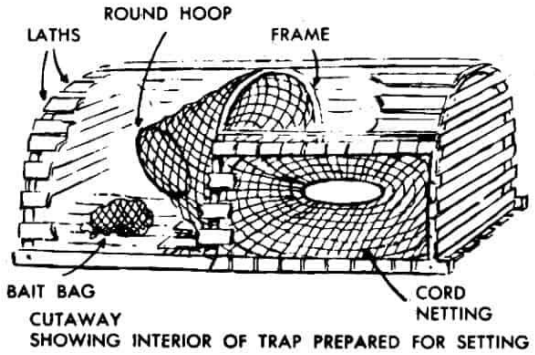
- Disarm a series of terminal-controlled traps to find lost treasure
- There are 4 traps in your file, each more tricky to disarm than the next
- Each trap has a special input string which will disable it. Your goal is to find those strings.
- Each user’s set of traps is unique! Your traps file will not be the same as your friend’s.

Important! Collaboration Policy Changes

- You may not share any part of your traps file with another student
- You may not discuss solutions, even at a high level
- You may not assist other students with disarming their traps
- TAs will also have a different TA hours policy:
  - We will be happy to help you with GDB (ie print vs x/s, setting breakpoints, etc), general approaches to the problem, and specific assembly questions.
  - However, TAs will NOT look at your assembly code, will not help with understanding chunks of code (ie what does this block of assembly code do), and will not give hints on the puzzles.

Project Roadmap

- Get your unique traps file by running the command `cs0330_install traps`
- Run the executable traps file in gdb, stepping through the assembly instructions.
  - Pay attention to register values while you do this! You're on the lookout for clues to help you create inputs which will disarm the trap, one stage at a time.
- As you disarm each level, save your answers into a text file.



Tools to Use

- `gdb`: Use gdb to step through the program, following its execution and disassembling functions as you go. Make sure to keep the cs033 gdb guide and x86 guide handy. You can find both on the front page of the course site.
- `objdump -t`: A good starting point. Gives the names and addresses of all of the functions in the trap. You can use these function names to set breakpoints.
- `objdump -d`: Provides a full disassembly of the trap. You may find it useful to redirect its output to a file and keep notes in that file as you step through the program.
- `strings`: Prints out all strings in the program that are more than 3 characters long.

Tips

- `gdb` can provide information about the state of the registers
  - type `info registers` or `i r`
- Use `print` and `x/s` to see what's stored in registers
- `si` is a gdb command which enables you to step over a single x86-64 assembly instruction
- Keep the gdb cheatsheet and the x86-64 cheatsheet at hand when working through the assembly code
- Use gdb's `TUI` mode to see all the relevant info you need at the same time
  - `TUI enable` to turn on the Text User Interface (TUI) mode
  - `Layout reg` to add a window which shows what info is in the registers
  - `Layout asm` to add a window which shows the relevant assembly code
  - `Refresh` to refresh the windows in case of bugs

Hands-on Example

Questions?

Example code from the demo!

Check out the following code if you want to further review the demo we just went over! The C code and assembly code from level_0 are both included.

```
0000000000001351 <level_zero>:
    1351:       55                  push   %rbp
    1352:       48 89 e5                mov    %rsp,%rbp
    1355:       48 83 ec 20             sub    $0x20,%rsp
    1359:       48 89 7d e8             mov    %rdi,-0x18(%rbp)
    135d:       48 8b 55 e8             mov    -0x18(%rbp),%rdx
    1361:       48 8d 45 f8             lea    -0x8(%rbp),%rax
    1365:       48 89 d6                mov    %rdx,%rsi
    1368:       48 89 c7                mov    %rax,%rdi
    136b:       e8 ba ff ff ff      callq  132a <read_one_int>
    1370:       8b 45 f8                mov    -0x8(%rbp),%eax
    1373:       83 f8 09                cmp    $0x9,%eax
    1376:       7f 18               jg     1390 <level_zero+0x3f>
    1378:       48 8b 45 e8             mov    -0x18(%rbp),%rax
    137c:48 89 c6                   mov    %rax,%rsi
    137f: bf 00 00 00 00            mov    $0x0,%edi
    1384:       e8 2c fe ff ff      callq  11b5 <pop_trap>
    1389:       b8 00 00 00 00          mov    $0x0,%eax
    138e:       eb 46               jmp    13d6 <level_zero+0x85>
    1390: c7 45 fc 00 00 00 00          movl   $0x0,-0x4(%rbp)
    1397: eb 0d                 jmp    13a6 <level_zero+0x55>
    1399: 8b 45 f8              mov    -0x8(%rbp),%eax
    139c: 83 c0 02              add    $0x2,%eax
    139f:  89 45 f8             mov    %eax,-0x8(%rbp)
    13a2: 83 45 fc 01           addl   $0x1,-0x4(%rbp)
    13a6: 83 7d fc 04           cmpl   $0x4,-0x4(%rbp)
    13aa: 7e ed                 jle    1399 <level_zero+0x48>
    13ac: 8b 55 f8              mov    -0x8(%rbp),%edx
    13af:  8b 05 cb 30 00 00            mov    0x30cb(%rip),%eax        # 4480 <SECRET_INT>
    13b5: 39 c2                 cmp    %eax,%edx
    13b7: 74 18                 je     13d1 <level_zero+0x80>
    13b9: 48 8b 45 e8               mov    -0x18(%rbp),%rax
    13bd: 48 89 c6              mov    %rax,%rsi
    13c0: bf 00 00 00 00            mov    $0x0,%edi
    13c5: e8 eb fd ff ff        callq  11b5 <pop_trap>
    13ca: b8 00 00 00 00            mov    $0x0,%eax
    13cf:  eb 05                jmp    13d6 <level_zero+0x85>
    13d1: b8 01 00 00 00            mov    $0x1,%eax
    13d6: c9                    leaveq
    13d7: c3                    retq
```

```
int level_zero(char *input) {
	int n;
	read_one_int(&n, input);

	if (n < 10) {
		pop_trap(0, input);
		return 0;
	}


	for(int i = 0; i < 5; i++) {
		n += 2;
	}

	if (n != SECRET_INT) {
		pop_trap(0, input);
		return 0;
	}

	return 1;
}
```