# Debuggers

Arvind S Raj
(arvindsraj@am.amrita.edu)

16SN715 Introduction to Software Reverse Engineering

M.Tech CSN Jan-May 2018

# Introduction

- Debugging assembly programs is hard: printf approach doesn't scale.
- Debuggers and few other CLI tools simplify task greatly.
- Agenda: Discuss few tools and some available extensions that simplify debugging programs.
- Will be useful for your assignments hopefully!

# What are debuggers?

- **Debuggers**: Allow inspecting process state during run-time.
- Hardware debuggers also exist. We discuss only software ones though. Eg: GDB.
- Enable stepping through each instruction in binary enabling close monitoring of process.
- Some also allow scripting during run-time: write programs to analyse the process!

# Outline

- ltrace for initial debugging.
- GNU debugger for more effective debugging of assembly programs.
- peda: GDB configuration to improves usage experience.
- Other GDB configuration and even front-ends exist! Feel free to experiment to find what you like.

# ltrace

- Library call tracer: Prints out dynamic library functions called along with their arguments.
- Useful to determine if library calls you make are not working as expected.
- Usage: **ltrace** $< /path/to/binary >$.
- Provides more capabilities and highly configurable. Read man page of ltrace and ltrace.conf.

# GNU Debugger

- Debugger in GNU's binary utilities. Most popular Linux debugger.
- **peda**: GDB utility aimed at exploit development but useful for reverse engineering too. Improves GDB's user friendliness.
- Uses several third party libraries to improve debugging experience.

# GDB trivia

- Default AT&T syntax. Prefer Intel syntax: **set disassembly-flavor intel**. peda uses this by default.
- Heavily configurable using **$HOME/.gdbinit**.
- Support basic scripting functionality.
- peda adds a Python scripting interface - get full power of Python libraries!

# Breakpoints

- **Breakpoint**: Pausing program execution at a point. Very useful for debugging if you know approximately where bug is caused.
- **Types**: Software and hardware. Latter requires hardware support and typically faster but only limited number possible.
- **b[reak] [addr]**: Set breakpoint at *addr*. If no address is specified, set breakpoint at next instruction to be executed.

# Breakpoints(cont.)

- **i[nfo] b[reakpoints]**: View current breakpoints.
- **dis[able] [num]**: Disable an active breakpoint. No argument $\implies$ disable all.
- **en[able] [num]**: Enable a disabled breakpoint. No argument $\implies$ enable all.
- **ig[nore]** $<$**num**$>$ $<$**count**$>$: Ignore breakpoint *num count* times.

# Breakpoints(cont.)

- **del[ete] [num]**: Delete breakpoint *num*, if exists. No argument $\implies$ delete all breakpoints.
- **tb[reak] [addr]**: Similar to *break* except breakpoint is deleted after one hit.
- **hb[reak] [addr]**: Similar to *break* but creates hardware breakpoint only.
- **thb[reak] [addr]**: Combination of *tbreak* and *hbreak*.

# Conditional Breakpoints

- **Conditional breakpoints**: Stop execution at a breakpoint if a specific condition is satisfied.
- Useful for isolating specific cases you are interested in(eg: when a variable has value 10).
- Creating a conditional breakpoint: **b** <**addr**> **if** <**expression**>. eg: b main if $edi==1.
- Adding condition to existing breakpoint: **cond[ition]** <**num**> <**expression**>. Eg: condition 7 $edi==1.

# Watchpoints

- **Watchpoints**: Similar to breakpoints for memory - interrupt execution if a particular memory location is accessed.
- Useful for monitoring changes made to variables in memory.
- **wa[tch]** <**addr**>: Interrupt if *addr* is written to.
- **rw[atch]** <**addr**>: Interrupt if *addr* is read.
- **awa[tch]** <**addr**>: Interrupt if *addr* is read or written to.

# Stepping through instructions

- **s[tep]i [num]**: Execute *num*(default: 1) instruction and stop at next. If current instruction is call, stop at first instruction of the function called.
- **n[ext]i [num]**: Similar to stepi except that if current instruction is call, the entire function is executed. Useful for skipping library functions.
- **fin[ish]**: Complete executing current function.

# Stepping through instructions

- **c[ontinue]**: Continue till next breakpoint/end of program.
- **nextjmp**: peda command. Continue till next jmp instruction is encountered. Ignores breakpoints on the way.
- **nextc[all]**: peda command. Continue till next call instruction is encountered. Ignores breakpoints on the way.

# Inspecting memory

- **x**: Display contents of memory in multiple formats(eg: binary), sizes(1, 2, 4 and 8 bytes) and types(eg: string).
- Eg: **x/10xg $edx**: Read 10 64-bit values from address *$edx* and print them as hex values.
- **tel[escope]**: peda command. Display contents and dereferences them too. Eg: **tel $edx 10**.
- Program stack is also memory - above commands can be used and **stack**/**context stack**(peda).

# Inspecting CPU registers

- **i[nfo] r[egisters]**: Display value of **all** registers.
- **i r $ecx $eax $ebx**: Display values of registers ecx, eax and ebx.
- **context register**: peda command. Display values of GPRs(EAX-EDX, EBP, ESP, EDI, ESI, EIP) and interpret contents(eg: dereference if pointer).
- Modify value of registers: **set $eax = 1**.

# Viewing assembly instructions in binary

- **x/[0-9]+i** <**address**>. View N instructions starting at *address*. Registers, variables also allowed.
- **context code**: peda command; displays next few instructions that will be executed by evaluating conditional instructions, if any.
- **disas[semble] [address]**: View all instructions in a function. No argument $\Longrightarrow$ current function.
- **pdisass** <**address**>: peda command. Similar to disas but coloured, nicer output.

## Hands on activity!

# Use GDB to find the problem with the given assembly programs and thus, fix them too.

# GDB

- Discussed few features: many more features exist but not important here.
- Has several front-ends: pwndbg, GEF and even windowed DDD!
- Python scripting also supported.
- peda features too not discussed in detail. You've probably already done it.
- Topics we discussed: not exhaustive but sufficient to find out more information.