gdb: your new best friend

captainGeech

Agenda

- Basic usage
- Extensions
- pwntools

Basic Usage

Requirements

- Linux
  - macOS, use lldb
  - Windows, x64dbg or windbg if you are a sadist (or doing kernel debugging)
- Binary
  - Ideally: with debug symbols
  - Works fine with just function names
  - On stripped binaries, you'll want to have a proper static analysis tool
    - ida, ghidra, r2, etc.

.gdbinit

- gdb config file
- Will be automatically run when you launch GDB (similar to .zshrc, etc.)
- Recommended values:

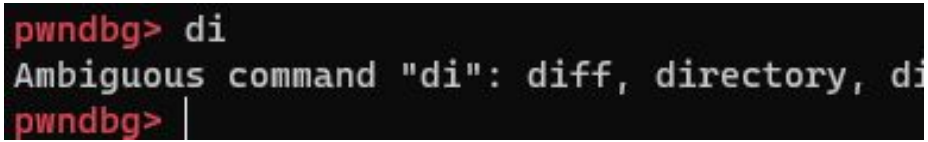
```
# geech @ ctf in ~/samurai-gdb using py3 [18:51:12]
$ cat ~/.gdbinit
       File: /home/geech/.gdbinit
   1   set disassembly-flavor att
   2   set follow-fork-mode child
   3   set auto-load safe-path /
   4   source ~/pwndbg/gdbinit.py
   5   #set detach-on-fork off
```

Usage

- To load a binary
  - `gdb [path to binary]`
  - shell rc: `alias gdb="gdb -q"`
- Run your binary
  - `run`
    - No stdin/argv
  - `run [arg1] [arg2] [...]`
    - argv
  - `run < [path to file]`
    - file to stdin
  - `run <<< asdf`
    - the bytes "asdf" will be on stdin

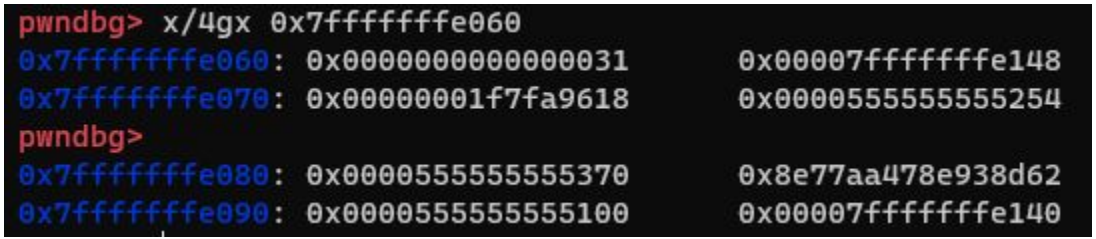
Command shortening

- All commands can be shortened, as long as they aren’t ambiguous
  - break -> b



Rerunning commands

- By pressing <Enter>, you can rerun the same command you ran previously
- For commands involving a memory instruction, it intelligently increments it



Control Flow

- `break *addr`
  - Set a breakpoint
  - e.g., break *0x401000
- `continue`
  - Continue execution
- `restart`
  - Restart the program, using the same arguments as before
- `si`
  - **s**tep **i**nstruction (step into)
- `ni`
  - **n**ext **i**nstruction (step over)
- Can also do `step` and `next` for source level debugging

Calling Functions

- You can make arbitrary function calls in gdb!
- Need to specify the return type of the function if there isn't DWARF data

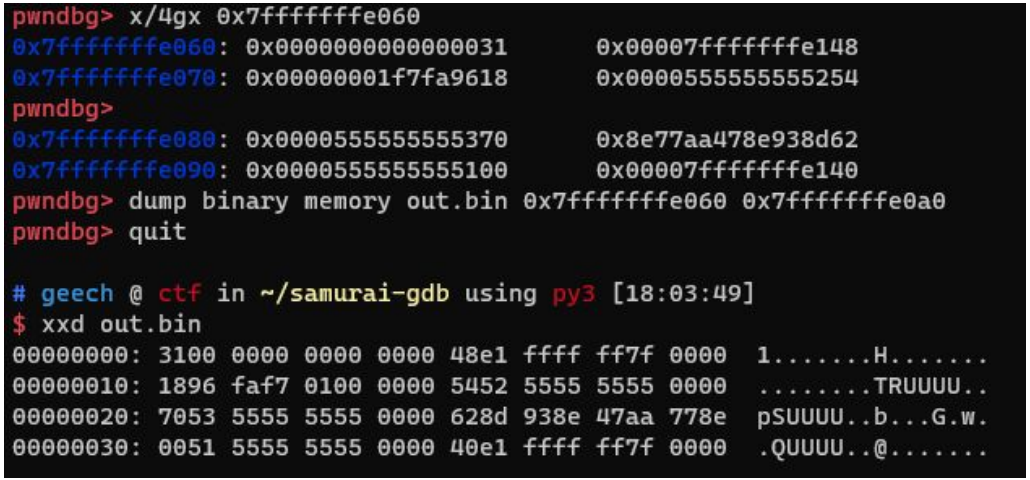
```
pwndbg> call (void *)malloc(0x10)
$1 = (void *) 0x7ffff7fb4f80
pwndbg> call (void)free($1)
pwndbg>
```

Viewing Memory

- x
  - Examine memory
  - x/[count][width][format]
    - The order of these doesn't matter
- Examples
  - `x/8gx 0x1000`
    - 8 qwords (**g**iant words) in hex format starting at 0x1000
  - `x/s 0x1000`
    - An ASCII string at 0x1000
  - `x/4i 0x1000`
    - 4 instructions starting at 0x1000
- https://sourceware.org/gdb/current/onlinedocs/gdb/Memory.html

Dumping Memory

- `dump binary memory [filename] [start addr] [end addr]`
  - Save the raw bytes from [start addr] to [end addr] to [filename]
  - More options available, but 99% of the time, that's what you want
  - https://sourceware.org/gdb/onlinedocs/gdb/Dump_002fRestore-Files.html



Editing Memory

- You can directly edit any bytes in memory, regardless of memory page permissions
  - `set *((char *) 0x1000) = 0xff`
    - Set the byte at 0x1000 to 0xff
  - Cast & dereference a pointer to the correct width you wish to write
    - `*((char *) 0x1000)` for one byte, `*((short *)0x1000)` for two bytes, etc.

```
pwndbg> call (void *)malloc(0x10)
$2 = (void *) 0x7ffff7fb4f80
pwndbg> x/gx $2
0x7ffff7fb4f80: 0x0000000000000000
pwndbg> set *((char *) $2) = 0xff
pwndbg> x/gx $2
0x7ffff7fb4f80: 0x00000000000000ff
pwndbg>
```

Viewing Registers

- info reg
  - Shows all of the common registers (no FPU, xmm/ymm, etc)
- info all-registers
  - Shows **every** register (use with caution)

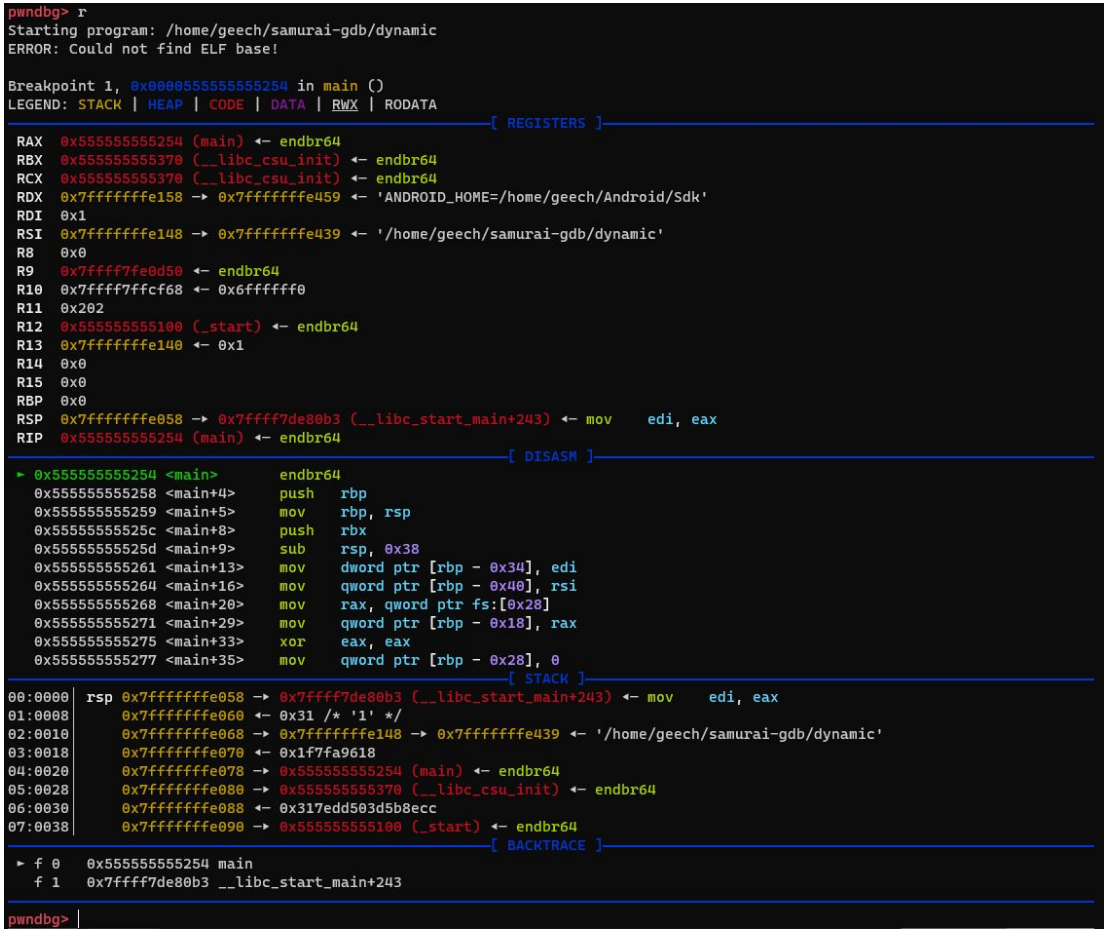
```
pwndbg> i r
rax            0x555555555254      93824992236116
rbx            0x555555555370      93824992236400
rcx            0x555555555370      93824992236400
rdx            0x7fffffffe158      140737488347480
rsi            0x7fffffffe148      140737488347464
rdi            0x1                 1
rbp            0x0                 0x0
rsp            0x7fffffffe058      0x7fffffffe058
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7ffff7ffcf68      140737354125160
r11            0x202               514
r12            0x555555555100      93824992235776
r13            0x7fffffffe140      140737488347456
r14            0x0                 0
r15            0x0                 0
rip            0x555555555254      0x555555555254 <main>
eflags         0x246               [ PF ZF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

Extensions

make gdb not bad

- Vanilla GDB is trash
- Fabulous people have used the Python API to make it better
  - https://github.com/pwndbg/pwndbg
  - https://github.com/hugsy/gef
- I prefer pwndbg but they are fairly similar

context


pwndbg> r
Starting program: /home/geech/samurai-gdb/dynamic
ERROR: Could not find ELF base!
Breakpoint 1, 0x0000555555555254 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[REGISTERS]
RAX 0x555555555254 (main) ◂— endbr64
RBX 0x555555555370 (__libc_csu_init) ◂— endbr64
RCX 0x555555555370 (__libc_csu_init) ◂— endbr64
RDX 0x7fffffffe158 —▸ 0x7fffffffe459 ◂— 'ANDROID_HOME=/home/geech/Android/Sdk'
RDI 0x1
RSI 0x7fffffffe148 —▸ 0x7fffffffe439 ◂— '/home/geech/samurai-gdb/dynamic'
R8 0x0
R9 0x7ffff7fe0d50 ◂— endbr64
R10 0x7ffff7ffcf68 ◂— 0x6ffffff0
R11 0x202
R12 0x555555555100 (_start) ◂— endbr64
R13 0x7fffffffe140 ◂— 0x1
R14 0x0
R15 0x0
RBP 0x0
RSP 0x7fffffffe058 —▸ 0x7ffff7de80b3 (__libc_start_main+243) ◂— mov edi, eax
RIP 0x555555555254 (main) ◂— endbr64
[DISASM]
▸ 0x555555555254 <main> endbr64
0x555555555258 <main+4> push rbp
0x555555555259 <main+5> mov rbp, rsp
0x55555555525c <main+8> push rbx
0x55555555525d <main+9> sub rsp, 0x38
0x555555555261 <main+13> mov dword ptr [rbp - 0x34], edi
0x555555555264 <main+16> mov qword ptr [rbp - 0x40], rsi
0x555555555268 <main+20> mov rax, qword ptr fs:[0x28]
0x555555555271 <main+29> mov qword ptr [rbp - 0x18], rax
0x555555555275 <main+33> xor eax, eax
0x555555555277 <main+35> mov qword ptr [rbp - 0x28], 0
[STACK]
00:0000 rsp 0x7fffffffe058 —▸ 0x7ffff7de80b3 (__libc_start_main+243) ◂— mov edi, eax
01:0008 0x7fffffffe060 ◂— 0x31 /* '1' */
02:0010 0x7fffffffe068 —▸ 0x7fffffffe148 —▸ 0x7fffffffe439 ◂— '/home/geech/samurai-gdb/dynamic'
03:0018 0x7fffffffe070 ◂— 0x1f7fa9618
04:0020 0x7fffffffe078 —▸ 0x555555555254 (main) ◂— endbr64
05:0028 0x7fffffffe080 —▸ 0x555555555370 (__libc_csu_init) ◂— endbr64
06:0030 0x7fffffffe088 ◂— 0x317edd503d5b8ecc
07:0038 0x7fffffffe090 —▸ 0x555555555100 (_start) ◂— endbr64
[BACKTRACE]
▸ f 0 0x555555555254 main
f 1 0x7ffff7de80b3 __libc_start_main+243
pwndbg>

Other pwndbg features

- Better support for QEMU remote debugging
- IDA Pro/Ghidra integration
- Memory map
- Searching for pointers, strings, etc.
- windbg compatibility
- https://github.com/pwndbg/pwndbg/blob/dev/FEATURES.md

vmmap

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
    0x555555554000     0x555555555000 r--p     1000 0      /home/geech/samurai-gdb/dynamic
    0x555555555000     0x555555556000 r-xp     1000 1000   /home/geech/samurai-gdb/dynamic
    0x555555556000     0x555555557000 r--p     1000 2000   /home/geech/samurai-gdb/dynamic
    0x555555557000     0x555555558000 r--p     1000 2000   /home/geech/samurai-gdb/dynamic
    0x555555558000     0x555555559000 rw-p     1000 3000   /home/geech/samurai-gdb/dynamic
    0x7ffff7dc1000     0x7ffff7de6000 r--p    25000 0      /usr/lib/x86_64-linux-gnu/libc-2.31.so
    0x7ffff7de6000     0x7ffff7f5e000 r-xp   178000 25000  /usr/lib/x86_64-linux-gnu/libc-2.31.so
    0x7ffff7f5e000     0x7ffff7fa8000 r--p    4a000 19d000 /usr/lib/x86_64-linux-gnu/libc-2.31.so
    0x7ffff7fa8000     0x7ffff7fa9000 ---p     1000 1e7000 /usr/lib/x86_64-linux-gnu/libc-2.31.so
    0x7ffff7fa9000     0x7ffff7fac000 r--p     3000 1e7000 /usr/lib/x86_64-linux-gnu/libc-2.31.so
    0x7ffff7fac000     0x7ffff7faf000 rw-p     3000 1ea000 /usr/lib/x86_64-linux-gnu/libc-2.31.so
    0x7ffff7faf000     0x7ffff7fb5000 rw-p     6000 0      anon_7ffff7faf
    0x7ffff7fc9000     0x7ffff7fcd000 r--p     4000 0      [vvar]
    0x7ffff7fcd000     0x7ffff7fcf000 r-xp     2000 0      [vdso]
    0x7ffff7fcf000     0x7ffff7fd0000 r--p     1000 0      /usr/lib/x86_64-linux-gnu/ld-2.31.so
    0x7ffff7fd0000     0x7ffff7ff3000 r-xp    23000 1000   /usr/lib/x86_64-linux-gnu/ld-2.31.so
    0x7ffff7ff3000     0x7ffff7ffb000 r--p     8000 24000  /usr/lib/x86_64-linux-gnu/ld-2.31.so
    0x7ffff7ffc000     0x7ffff7ffd000 r--p     1000 2c000  /usr/lib/x86_64-linux-gnu/ld-2.31.so
    0x7ffff7ffd000     0x7ffff7ffe000 rw-p     1000 2d000  /usr/lib/x86_64-linux-gnu/ld-2.31.so
    0x7ffff7ffe000     0x7ffff7fff000 rw-p     1000 0      anon_7ffff7ffe
    0x7ffffffde000     0x7ffffffff000 rw-p    21000 0      [stack]
0xffffffffff600000 0xffffffffff601000 --xp     1000 0      [vsyscall]
```

search

```
pwndbg> search -s "visitor"
dynamic         0x555555556018 'visitor #%d\n'
dynamic         0x555555557018 'visitor #%d\n'
pwndbg> search -s "/bin/sh"
libc-2.31.so    0x7ffff7f785aa 0x68732f6e69622f /* '/bin/sh' */
```

pwntools

pwntools + gdb = pwnage

You can run GDB commands and automatically attach with your pwntools script:

```
context.terminal = ['tmux', 'splitw', '-v']
gdb.attach(p, "b *0x400767")
```

https://docs.pwntools.com/en/stable/gdb.html

Demo