






Introducing GEF (GDB Enhanced Features)

Chris Alladoun ( _hugsy_ ||  hugsy)

About me (briefly)

- I'm Chris
 - a.k.a (*crazy rabbidz*) hugsy [<mailto:hugsy@blah.cat>]
 - Security Researcher
- Passion for low-level software stuff
 - Bug hunting, reversing
 - Dynamic & static analysis
 - Playing with weird architectures
 - Building tools around that to help me find bugs
- CTF player (team )

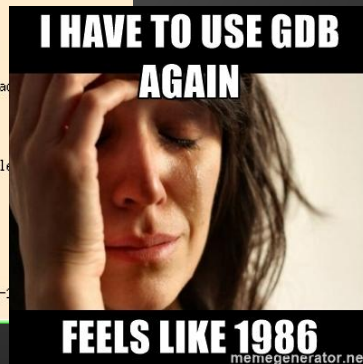


Why ?

- GDB (initial release = 1986!!) is the “*de facto*” debugger for all Linux/Unix systems
- Command line oriented (Unix philosophy)
- For debugging and/or exploit development, only has a fixed set of commands
- We're in 2017, and GDB is very (very!) far behind WinDBG (in terms of functionality)

```
GNU gdb 6.1.1 [FreeBSD]
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-marcel-freebsd"...
(gdb) b main
Breakpoint 1 at 0x80491d0: file locale.c, line 228.
(gdb) r
Starting program: /src/FreeBSD/svn/head/usr.bin/locale/locale

Breakpoint 1, main (argc=Error accessing memory address 0x2: Bad
) at locale.c:228
228 {
(gdb) l
223     const char *boguslocales[] = { "UTF-8" };
224     #define NBOGUS (sizeof(boguslocales)/sizeof(boguslocale
225
226     int
227     main(int argc, char *argv[])
228     {
229         int    ch;
230         int    tmp;
231
232         while ((ch = getopt(argc, argv, "ackms:")) != -1
(gdb) █
```



Why ?

GDB embeds only its own “scripting” language

- Pretty good for quick’n dirty scripting
- But terrible for robust programming, developing new commands

Luckily, GDB 7.0 (2011) came with a nice surprise → a Python API !

- Which lead to a rise of cool new plugins (PEDA, Voltron, gdb-heap, !exploitable, etc.)



GDB Enhanced Features (a.k.a. GEF)

Pronounced “*Jeff*”

- 1st version started end 2011: mix of horrible scripts to learn ARM & MIPS and GDB-Python
 - Not at all for x86 though...
- (Current) 2nd version started in 2013: full rewrite with a proper abstraction layer to make commands work on any architecture supported by GDB → **GEF** was born !

Set of commands that extend **GDB** via its Python API

→ i.e. Requires GDB v7+ (compiled with Python{2,3} support)

Aims to be used

- For Exploit Dev & Reversers
- For Low-Level Developers too !

GEF

From Day-1, GEF was built and driven following those constraints

- Simple & fast to install / update (Plug'n Play)
- No dependency required (battery-included, no need for git/gcc/pip)
- Python2 / Python3 compatible
- CPU architecture agnostic
 - Currently supported:
 - x86 , x86-64 , ARM , MIPS , AARCH64 , PowerPC , SPARC , PPC64 , ...
 - But new architectures can be added in minutes
- Easily extensible
- Well documented

GEF

- Built by curiosity,



Started as a bunch of scattered command scripts to learn ARM

Improved by CTFs,



Provide fast and comprehensive access to info (registers, memory, etc.)

Perfect by community



Released as FOSS (MIT), many contributors, active IRC.

- Well-detailed documentation, systematically updated
- Many commands/aliases were inspired by WinDBG
- Aliased commands from PEDA for smooth transition :)
- Compatible with PwnTools

GEF coolest features

Demo: Basic use (context, vmmap, xinfo, hexdump, patch string,...)

Video: 1. GEF 101

```
gef> vmmap
Start      End      Offset  Perm Path
0x0000000004000000 0x0000000004010000 r-x /root/bof-aarch64
0x0000000004100000 0x0000000004110000 r-x /root/bof-aarch64
0x0000000004110000 0x0000000004120000 rwx /root/bof-aarch64
0x0000000004120000 0x0000000004130000 r-x /lib/aarch64-linux-gnu/libc-2.19.so
0x0000000004130000 0x0000000004140000 r-x /lib/aarch64-linux-gnu/libc-2.19.so
0x0000000004140000 0x0000000004150000 rwx /lib/aarch64-linux-gnu/libc-2.19.so
0x0000000004150000 0x0000000004160000 r-x /lib/aarch64-linux-gnu/libc-2.19.so
0x0000000004160000 0x0000000004170000 rwx /lib/aarch64-linux-gnu/libc-2.19.so
0x0000000004170000 0x0000000004180000 r-x [vdso]
0x0000000004180000 0x0000000004190000 r-x /lib/aarch64-linux-gnu/libc-2.19.so
0x0000000004190000 0x00000000041a0000 rwx /lib/aarch64-linux-gnu/libc-2.19.so
0x00000000041a0000 0x00000000041b0000 rwx [stack]
```

```
gef> hexdump byte $sp l64
0x00007fffffe230  40 05 40 00 00 00 00 b1 a2 a5 f7 ff 7f 00 00  @.....
0x00007fffffe240  00 00 04 00 00 00 00 18 e3 ff ff 7f 00 00  .....
0x00007fffffe250  88 b2 b9 f7 01 00 00 10 05 40 00 00 00 00  .....@.....
0x00007fffffe260  00 00 00 00 00 00 00 75 a3 d7 5a 38 df c7 c4  .....U.Z8...

gef> hexdump qword $sp l3
0x7fffffe230+0000 0x000000000400540
0x7fffffe230+0008 0x00007ffff7a5a2b1
0x7fffffe230+0010 0x000000000400000
gef>
```

```
Breakpoint 1, 0x000000000400f54 in main ()
gef> dereference $rdx
0x00007fffffe468 +0x0000: 0x00007fffffe7c5 → "XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat[...]"
gef>
```

```
[ registers ]
$rax : 0x0000000000002010 → 0x00
$rbx : 0x0000000000000000 → 0x100000000
$rcx : 0x00007fffffd1b20 → 0x100000000
$rdi : 0x0000000000002010 → 0x00
$rsi : 0x00007fffffe530 → 0x00007fffffe618 → "/home/ubuntu/malloc-test"
$rbp : 0x00007fffffe530 → 0x000000000400620 → "_libc_csu_init@0" push r15
$rdi : 0x0000000000000020 → 0x00
$rdi : 0x0000000000002010 → 0x00
$rip : 0x0000000000040054f → <main+4> call 0x4004a0 <realloc@plt>
$ra : 0x0000000000002000 → 0x00
$rsi : 0x0000000000000000 → 0x00
$rsi : 0x00007fffffd1b20 → 0x0000000000006020 → 0x00
$rsi : 0x0000000000000000 → 0x00
$rsi : 0x000000000004004c → <_start@0> xor edi, edi
$rsi : 0x00007fffffe610 → 0x01
$rsi : 0x0000000000000000 → 0x00
$rsi : 0x0000000000000000 → 0x00
$eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow resume virtualx86 identification]

[ stack ]
0x00007fffffe510 +0x00: 0x00007fffffe618 → 0x00007fffffe628 → "/home/ubuntu/malloc-test" ← $rsp
0x00007fffffe510 +0x08: 0x1004004c0 → 0x00
0x00007fffffe520 +0x10: 0x00007fffffe610 → 0x01
0x00007fffffe528 +0x18: 0x0000000000006020 → 0x00
0x00007fffffe530 +0x20: 0x00000000000400620 → <_libc_csu_init@0> push r15 ← $rbp
0x00007fffffe538 +0x28: 0x00007ffff7a2e830 → <_libc_start_main@240> mov edi, edi
0x00007fffffe540 +0x30: 0x00
0x00007fffffe540 +0x38: 0x00007fffffe618 → 0x00007fffffe628 → "/home/ubuntu/malloc-test"

[ code:lib386:x86-64 ]
0x4005ca <main+20> call 0x400490 <malloc@plt>
0x4005cf <main+25> mov QWORD PTR [rbp-0x8], rax
0x4005d3 <main+29> mov rax, QWORD PTR [rbp-0x8]
0x4005d7 <main+33> mov esi, 0x20
0x4005dc <main+38> mov rdi, rax
→ 0x4005df <main+41> call 0x4004a0 <realloc@plt>
0x4004a0 <realloc@plt+0> jmp QWORD PTR [rip+0x200b8a] # 0x601030
0x4004a6 <realloc@plt+6> push 0x3
0x4004ab <realloc@plt+11> jmp 0x400460
0x4004b0 jmp QWORD PTR [rip+0x200b42] # 0x600f78
0x4004b6 xchg ax, ax
0x4004b8 add BYTE PTR [rax], al

[ source:malloc-test.c:20 ]
16 /* printf("%p\n", ptr); */
17
18 // realloc
19 ptr = malloc(0x10);
// ptr=0x00007fffffe528 → [...] → 0x00
→ 20 realloc(ptr, 0x20);
21 realloc(ptr, 0x40);
22 realloc(ptr, 128*1024);
23 free(ptr);
24

[ threads ]
[#0] Id 1, Name: "malloc-test", stopped, reason: SINGLE STEP

[ trace ]
[#0] RetAddr: 0x4005df, Name: main(argc=0x1, argv=0x7fffffe618)
```


GEF coolest features

Demo: Customizing GEF

Video: 2. Customizing GEF

GEF coolest features

Demo: Vulnerable format string detection & heap analysis

Video: 3. Runtime analysis with GEF

```

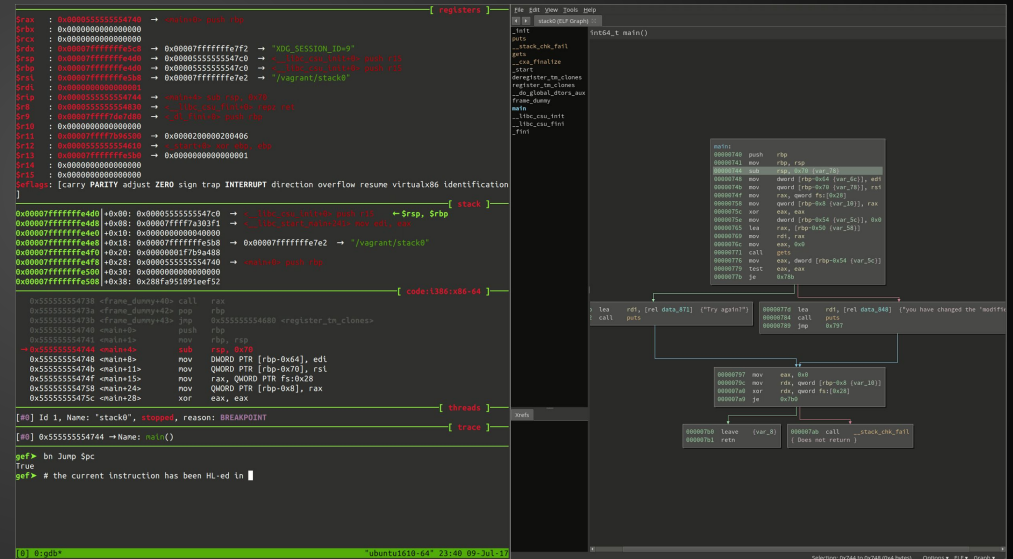
Description: =====[ Format String Detection ]=====
[+] Possible insecure format string '0x804c018' -> 0x804c018: 'b'aa\n'
[+] Triggered by 'printf()'
[+] Reason:
Call to 'printf()' with format string argument in position #0 is in
page 0x804c000 ([heap]) that has write permission
-----[regs]
$eax 0x0804c018 $ecx 0x0000000e $edx 0xf7fa977a $ebx 0x0804bda8
$esp 0xfffffd21c $ebp 0xfffffd238 $eip 0x00000000 $edi 0x00000000
$ebp 0xf7e53be0 $cs 0x00000023 $ss 0x0000002b $ds 0x0000002b
$es 0x0000002b $fs 0x00000000 $gs 0x00000063 $eflags [ PF SF IF ]
Flags: [ carry virtual PARITY adjust zero trap INTERRUPT direction overflow ]
-----[stack]
0xfffffd21c: 0x08048c27 -> leave          ← $sp
0xfffffd220: 0x0804c018 -> "aa\n"
0xfffffd224: 0x0804c008 -> "123"
0xfffffd228: 0xf7e58d1b -> call $fscanf@plt add ebx,0x1552e9
0xfffffd22c: 0xf7fae000 -> 0x1a6da8
0xfffffd230: 0x0
0xfffffd234: 0x0
0xfffffd238: 0xfffffd268 -> 0xfffffd298 -> 0x0
0xfffffd23c: 0x08048c99 -> add DWORD PTR [ebp-0xc],0x1
0xfffffd240: 0x0804b0a8 -> "aa"
-----[code:t386]
0xf7e53bd4: nop
0xf7e53bd5: xchg ax,ax
0xf7e53bd7: xchg ax,ax
0xf7e53bd9: xchg ax,ax
0xf7e53bdb: xchg ax,ax
0xf7e53bdd: xchg ax,ax
0xf7e53bdf: nop
0xf7e53be0: <printf> push ebx          ← $pc
0xf7e53be1: <printf+1> sub esp,0x18
0xf7e53be4: <printf+4> call 0xf7f2a18b
0xf7e53be9: <printf+9> add ebx,0x15a417
0xf7e53bef: <printf+15> lea eax,[esp+0x24]
-----[trace]
#0 0xf7e53be0 in printf () from /lib/t386-linux-gnu/t686/cmov/libc.so.6
#1 0x08048c27 in ?? ()
#2 0x08048c99 in ?? ()
#3 0x080487a2 in ?? ()
#4 0xf7e20a63 in __libc_start_main () from /lib/t386-linux-gnu/t686/cmov/libc.so.6

```

GEF coolest features

Demo: IDA Pro / Binary Ninja integration

Video: 4. Interfacing with IDA & BN



GEF coolest features

Demo: WinDBG ``dt`` command

Video: 5. Creating and Using Custom structure with GEF

GEF coolest features

Demo: Extending GEF

Video: 6. Extending GEF

GEF coolest features

For more goodies, install a few external dependencies to make GEF even greater:

- keystone-assemble : Assembly engine (requires ``keystone-engine`` PIP package)
- capstone-disassemble : Disassembly engine (requires ``capstone`` PIP package)
- unicorn-emulate : Emulation engine (requires ``unicorn`` PIP package)
- ropper : ROP Gadget generator (requires ``ropper`` PIP package)
- RetDec : Decompiler (requires ``retdec-python`` PIP3 package)

GEF coolest features

Demo: Using optional extensions

→ keystone-asm

Usage example: Write your own assembly directly from GDB! Why??? So you can write your own assembly directly in memory!

```
gef> cs $pc
→ 0x55555559430      xor    ebp, ebp
0x55555559432      mov    r9, rdx
0x55555559435      pop    rsi
0x55555559436      mov    rdx, rsp
0x55555559439      and    rsp, 0xfffffffffffff0

gef> asm xor ebp, ebp
[+] Assembling 1 instruction for X86:64 (little endian)
\x31\xed          # xor ebp, ebp
gef> asm -l $pc nop; nop
[+] Assembling 2 instructions for X86:64 (little endian)
[+] Overwriting 2 bytes at 0x000555555559430
gef> cs $pc
→ 0x55555559430      nop
0x55555559431      nop
0x55555559432      mov    r9, rdx
0x55555559435      pop    rsi
0x55555559436      mov    rdx, rsp
gef> 
```

GEF coolest features

Demo: Using optional extensions

- *keystone-assemble*
- **capstone-disassemble**

Usage example: ever needed to spot the difference between x86-64, x86-32 and x86-16? We got you covered.

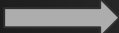
```
gef> cs $pc arch=X86 mode=64
→ 0x555555554610    xor    ebp, ebp
0x555555554612    mov    r9, rdx
0x555555554615    pop    rsi
0x555555554616    mov    rdx, rsp
0x555555554619    and    rsp, 0xfffffffffffffff0
gef> cs $pc arch=X86 mode=32
→ 0x555555554610    xor    ebp, ebp
0x555555554612    mov    r9, rdx
0x555555554615    pop    rsi
0x555555554616    mov    rdx, rsp
0x555555554619    and    rsp, 0xfffffffffffffff0
gef> cs $pc arch=X86 mode=16
→ 0x555555554610    xor    ebp, ebp
0x555555554612    mov    r9, rdx
0x555555554615    pop    rsi
0x555555554616    mov    rdx, rsp
0x555555554619    and    rsp, 0xfffffffffffffff0
gef>
```


GEF coolest features

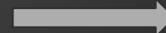
Demo: Using optional extensions

- ➔ *keystone-assemble*
- ➔ *capstone-disassemble*
- ➔ **unicorn-emulate**

Usage example: Too lazy to reverse a complex keygen function: no problem! Emulate it!



```
gef> unicorn-emulate -f 0x80485b1 -t 0x80486e0 -e /tmp/revbox.py
[*] Cannot copy page=0x7ffff7ff8000-0x7ffff7ffa000 : Cannot access memory at address 0x7ffff7ff8000
[+] Unicorn script generated as '/tmp/revbox.py'
gef>
```



```
1 #!/usr/bin/python -i
2
3 import unicorn
4
5 def emulate(eax):
6     emu = unicorn.Uc(unicorn.UC_ARCH_X86, unicorn.UC_MODE_32 + unicorn.UC_MODE_LITTLE_ENDIAN)
7     emu.reg_write(unicorn.x86_const.UC_X86_REG_EAX, eax)
8     emu.reg_write(unicorn.x86_const.UC_X86_REG_EBX, 0x0)
9     emu.reg_write(unicorn.x86_const.UC_X86_REG_ECX, 0x14d4658)
10    emu.reg_write(unicorn.x86_const.UC_X86_REG_EDX, 0x77fa3e4)
11    emu.reg_write(unicorn.x86_const.UC_X86_REG_ESP, 0xffffd220)
12    emu.reg_write(unicorn.x86_const.UC_X86_REG_EBP, 0xffffd240)
13    emu.reg_write(unicorn.x86_const.UC_X86_REG_ESI, 0x77fa6000)
14    emu.reg_write(unicorn.x86_const.UC_X86_REG_EDI, 0x77fa6000)
15    emu.reg_write(unicorn.x86_const.UC_X86_REG_EIP, 0x80485b1)
16    emu.reg_write(unicorn.x86_const.UC_X86_REG_EFLAGS, 0x202)
17    emu.men_map(0x0, 4096, 3)
18    emu.men_map(0x1000, 4096, 3)
19    emu.men_map(0x8048000, 4096, 5)
20    # Importing /home/vagrant/ctf/tokyo_western_ctf_2016/reverse_box: 0x8048000-0x8049000
21    data=open('/tmp/gef-0x8048000.raw', 'r').read()
22    emu.men_write(0x8048000, data)
23
24    emu.men_map(0x8049000, 4096, 3)
25    # Importing /home/vagrant/ctf/tokyo_western_ctf_2016/reverse_box: 0x8049000-0x804a000
26    data=open('/tmp/gef-0x8049000.raw', 'r').read()
27    emu.men_write(0x8049000, data)
28
29    emu.men_map(0x804a000, 4096, 3)
30    # Importing /home/vagrant/ctf/tokyo_western_ctf_2016/reverse_box: 0x804a000-0x804b000
31    data=open('/tmp/gef-0x804a000.raw', 'r').read()
32    emu.men_write(0x804a000, data)
33
34    emu.men_map(0xf7df7000, 1757184, 5)
35    # Importing /lib/x86_64-linux-gnu/libc-2.23.so: 0xf7df7000-0xf7fa000
36    data=open('/tmp/gef-0xf7df7000.raw', 'r').read()
37    emu.men_write(0xf7df7000, data)
38
39    emu.men_map(0xf7fa000, 8192, 1)
40    # Importing /lib/x86_64-linux-gnu/libc-2.23.so: 0xf7fa000-0xf7fa000
41    data=open('/tmp/gef-0xf7fa000.raw', 'r').read()
42    emu.men_write(0xf7fa000, data)
```

Practical example: <http://blahcat.github.io/2016/09/06/twctf-2016-reverse-box-writeup/>

GEF coolest features

Demo: Using optional extensions

- *keystone-assemble*
- *capstone-disassemble*
- *unicorn-emulate*
- **ropper**

Usage example: find ROP gadgets from the binary

```
gef> ropper --search 'pop %; pop %; ret'
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop %; pop %; ret

[INFO] File: /bin/ls
0x0000555555550000: pop r12; pop r13; pop r14; pop r15; pop rbp; ret;
0x0000555555550013: pop r12; pop r13; pop r14; pop r15; ret;
0x0000555555557ed0: pop r12; pop r13; pop r14; ret;
0x0000555555559c21: pop r12; pop r13; ret;
0x0000555555550d0a: pop r13; pop r14; pop r15; pop rbp; ret;
0x0000555555550b13: pop r13; pop r14; pop r15; ret;
0x0000555555557ed0: pop r13; pop r14; ret;
0x0000555555550d0a: pop r14; pop r15; pop rbp; ret;
0x0000555555550b17: pop r14; pop r15; ret;
0x0000555555550d0a: pop r15; pop rbp; ret;
0x0000555555550407: pop rbp; jmp rax; nop word ptr [rax + rax]; pop rbp; ret;
0x00005555555502b1e: pop rbp; pop r12; cmov rax, rdx; ret;
0x0000555555550a0c: pop rbp; pop r12; jmp 0x3860; nop dword ptr [rax + rax]; mov eax, 0xffffffff; ret;
0x0000555555557240: pop rbp; pop r12; jmp 0x5ed0; nop dword ptr [rax + rax]; mov eax, 0xffffffff; ret;
0x0000555555550b11: pop rbp; pop r12; pop r13; pop r14; pop r15; ret;
```

GEF coolest features

Demo: Using optional extensions

- *keystone-assemble*
- *capstone-disassemble*
- *unicorn-emulate*
- *ropper*
- **RetDec**

Or why not get the (pseudo) source code in C directly?

```

0x804850c      <main+12>      mov     edx,DWORD PTR [ebp+0x8]
0x804850f      <main+15>      mov     DWORD PTR [ebp-0x4],0x0
0x8048516      <main+22>      mov     DWORD PTR [ebp-0x8],edx

#0  0x08048506 in main ()

Temporary breakpoint 1, 0x08048506 in main ()
gef> decompile -s main
[+] Task submitted, waiting for decompilation to finish...
[+] Done
[+] Saved as '/tmp/tmp0a3ljigp.c'
#include <stdint.h>
int32_t entry_point(int32_t a1, int32_t a2, int32_t a3, int32_t a4);
int32_t unknown_8048480(void);
int32_t entry_point(int32_t a1, int32_t a2, int32_t a3, int32_t a4) {
    int32_t result = -1;
    if (a1 > 1) {
        unknown_8048480();
        result = 0;
    }
    return result;
}
gef>

```

Morale

That was just a small sample of what GEF can do!

- GEF is self-contained
 - ◆ But easily extensible
- It combines more than 50 commands to drastically improve GDB
- Feel free to contribute (pull request, bug report, etc.)
 - ◆ Got a cool idea for integrating a new command in GEF ? Let's talk !

The Python API of GDB is awesome

- Let's do more of it !

Together the FOSS community can bring GDB up to the 21st century

- ... and try to catch up with WinDBG

Special Thanks To ...

- All of the 11 (to this day) contributors on GitHub
 - Especially [@Grazfather](#)
- Beyond Security (SSD - SecuriTeam Secure Disclosure) for their SSD community sponsoring program (<https://www.beyondsecurity.com/ssd.html>)
- To ToolsWatch (<http://www.toolswatch.org>) for Black Hat Arsenal, and naturally to Black Hat
- And last but not least, to **YOU** for listening !
 - Enjoy Black Hat !



Links

GEF

- Project: <https://github.com/hugsy/gef>
- Docs: <https://gef.readthedocs.io/en/latest/>
- External scripts open repository: <https://github.com/hugsy/gef-scripts>
- External structures open repository: <https://github.com/hugsy/gef-structs>
- GEF Tutorials Channel on YouTube : <https://goo.gl/1QAZM4>
- IRC: [#gef](https://freenode.net)
- Various QEMU images embedding GEF: <https://blahcat.github.io/2017/06/25/qemu-images-to-play-with/>

Other valuable resources

- GDB Python API: <https://sourceware.org/gdb/onlinedocs/gdb/Python-API.html>
- fG!'s legendary gdbinit: <https://github.com/gdbinit/Gdbinit>
- Hi GDB, this is Python: <http://0vercl0k.tuxfamily.org/bl0g/?p=226>