Introducing GEF (GDB Enhanced Features)

Chris Alladoum (_hugsy_ || hugsy)

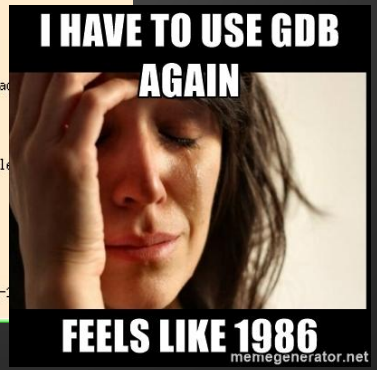
About me (briefly)

- 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 THE GOONIES)

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:")) != -
(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

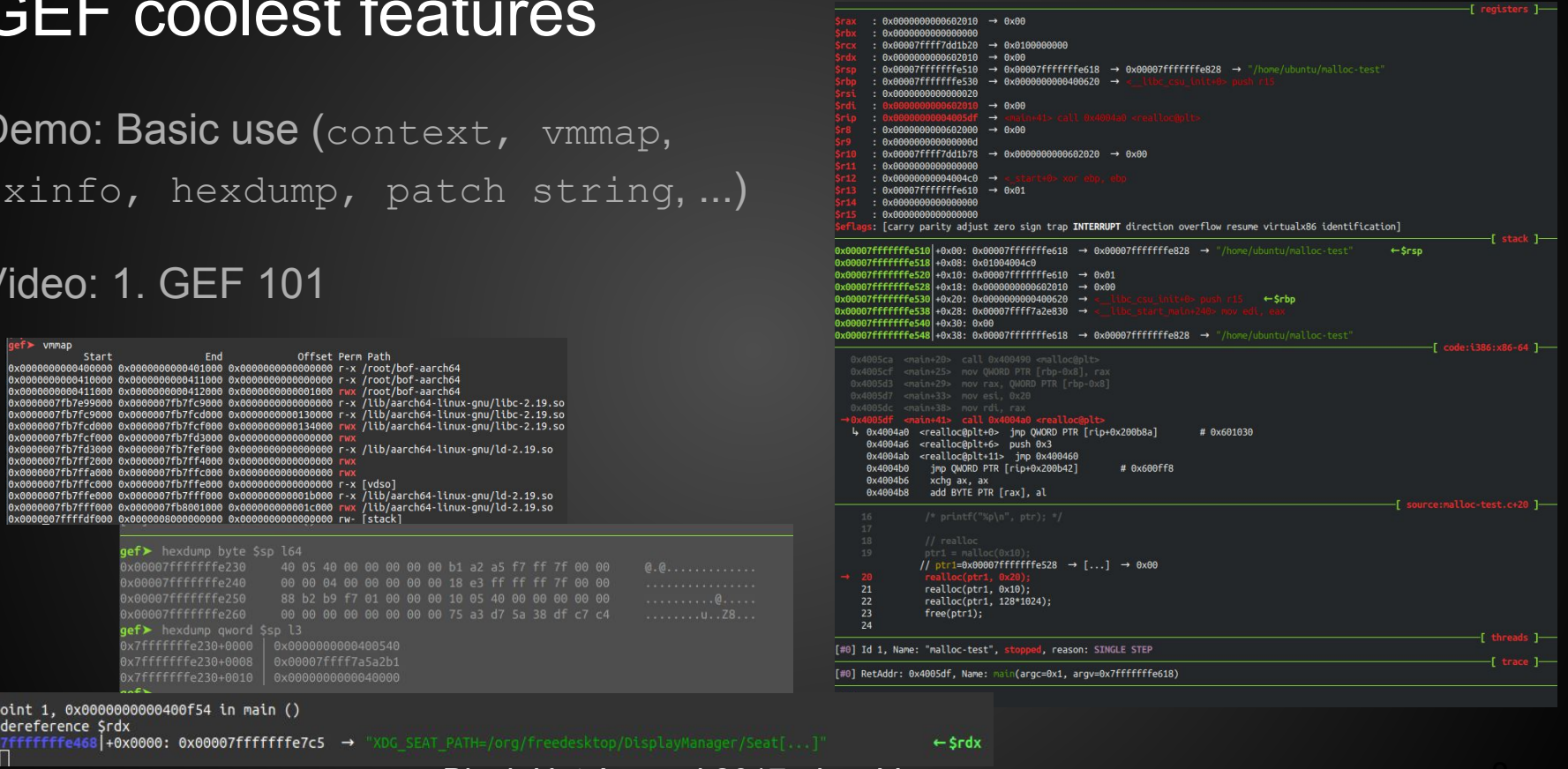
- Built by curiosity,
  - Started as a bunch of scattered command scripts to learn ARM
- Improved by CTFs,
  - Provide fast and comprehensive access to info (registers, memory, etc.)
- Perfected by community
  - Released as FOSS (MIT), many contributors, active IRC.

- 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 coolest features

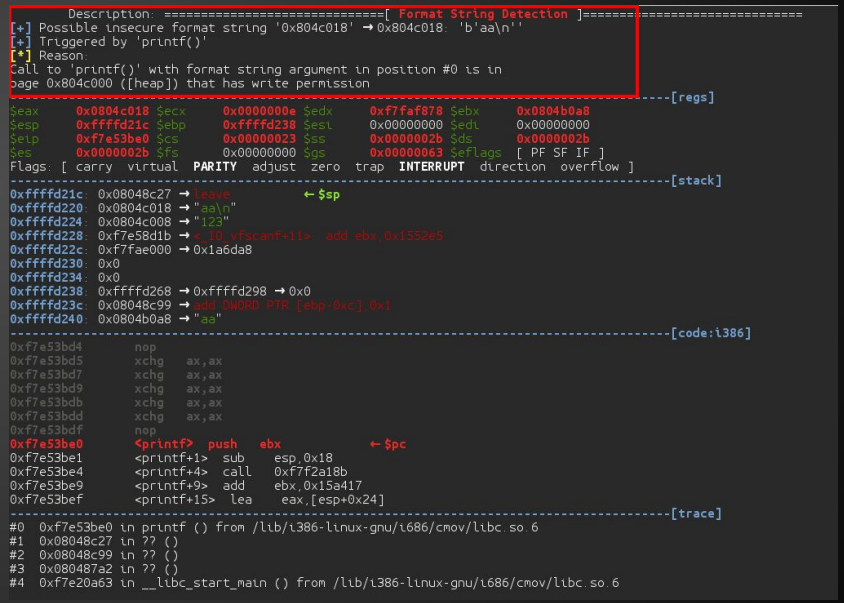
Demo: Customizing GEF

Video: 2. Customizing GEF

GEF coolest features

Demo: Vulnerable format string detection & heap analysis

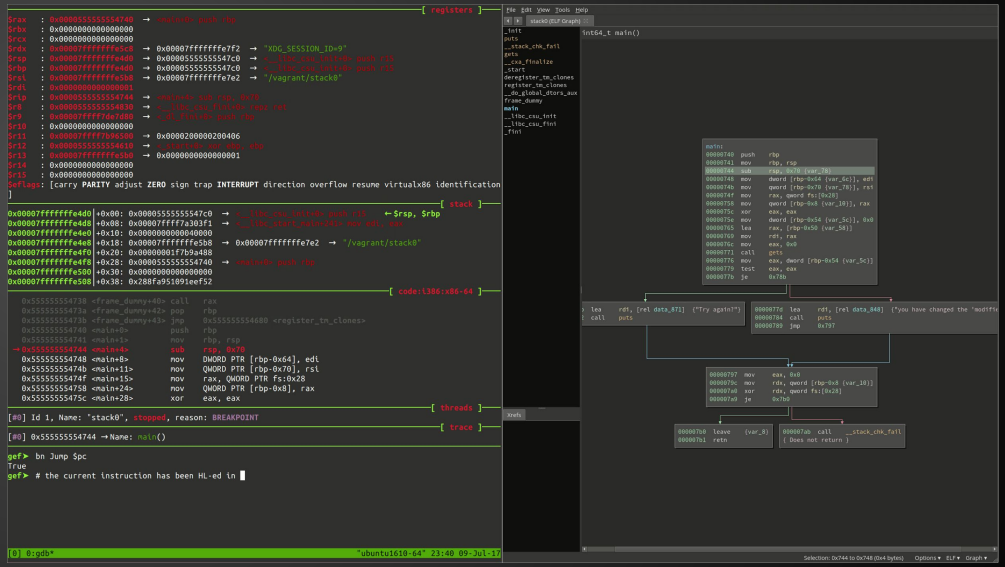
Video: 3. Runtime analysis with GEF



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-assemble**

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

```
gef➤  cs $pc
 →    0x555555559430                  xor    ebp, ebp
      0x555555559432                  mov    r9, rdx
      0x555555559435                  pop    rsi
      0x555555559436                  mov    rdx, rsp
      0x555555559439                  and    rsp, 0xfffffffffffffff0
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 0x0000555555559430
gef➤  cs $pc
 →    0x555555559430                  nop
      0x555555559431                  nop
      0x555555559432                  mov    r9, rdx
      0x555555559435                  pop    rsi
      0x555555559436                  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➤
```

```
#!/usr/bin/python -i

import unicorn

def emulate(eax):
    emu = unicorn.Uc(unicorn.UC_ARCH_X86, unicorn.UC_MODE_32 + unicorn.UC_MODE_LITTLE_ENDIAN)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_EAX, eax)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_EBX, 0x0)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_ECX, 0x14d4e658)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_EDX, 0xf7fa63e4)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_ESP, 0xffffd220)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_EBP, 0xffffd248)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_ESI, 0xf7fa6000)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_EDI, 0xf7fa6000)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_EIP, 0x80485b1)
    emu.reg_write(unicorn.x86_const.UC_X86_REG_EFLAGS, 0x202)
    emu.mem_map(0x0, 4096, 3)
    emu.mem_map(0x1000, 4096, 3)
    emu.mem_map(0x8048000, 4096, 5)
    # Importing /home/vagrant/ctf/tokyo_western_ctf_2016/reverse_box: 0x8048000-0x8049000
    data=open('/tmp/gef-0x8048000.raw', 'r').read()
    emu.mem_write(0x8048000, data)

    emu.mem_map(0x8049000, 4096, 1)
    # Importing /home/vagrant/ctf/tokyo_western_ctf_2016/reverse_box: 0x8049000-0x804a000
    data=open('/tmp/gef-0x8049000.raw', 'r').read()
    emu.mem_write(0x8049000, data)

    emu.mem_map(0x804a000, 4096, 3)
    # Importing /home/vagrant/ctf/tokyo_western_ctf_2016/reverse_box: 0x804a000-0x804b000
    data=open('/tmp/gef-0x804a000.raw', 'r').read()
    emu.mem_write(0x804a000, data)

    emu.mem_map(0xf7df7000, 1757184, 5)
    # Importing /lib/i386-linux-gnu/libc-2.23.so: 0xf7df7000-0xf7fa4000
    data=open('/tmp/gef-0xf7df7000.raw', 'r').read()
    emu.mem_write(0xf7df7000, data)

    emu.mem_map(0xf7fa4000, 8192, 1)
    # Importing /lib/i386-linux-gnu/libc-2.23.so: 0xf7fa4000-0xf7fa6000
    data=open('/tmp/gef-0xf7fa4000.raw', 'r').read()
    emu.mem_write(0xf7fa4000, 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

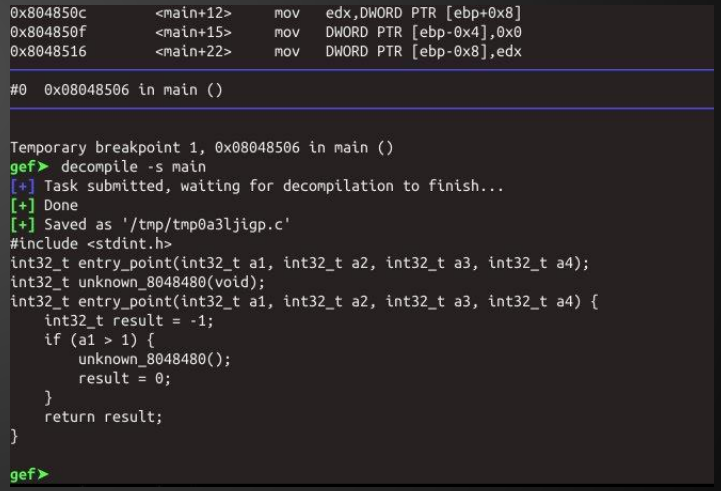
```
gef➤  ropper --search 'pop %; pop %; ret'
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop %; pop %; ret

[INFO] File: /bin/ls
0x0000555555556d88: pop r12; pop r13; pop r14; pop r15; pop rbp; ret;
0x0000555555558b13: pop r12; pop r13; pop r14; pop r15; ret;
0x0000555555555fed6: pop r12; pop r13; pop r14; ret;
0x0000555555559c21: pop r12; pop r13; ret;
0x0000555555556d88a: pop r13; pop r14; pop r15; pop rbp; ret;
0x0000555555558b15: pop r13; pop r14; pop r15; ret;
0x0000555555555fed8: pop r13; pop r14; ret;
0x0000555555556d68c: pop r14; pop r15; pop rbp; ret;
0x0000555555558b17: pop r14; pop r15; ret;
0x0000555555556d68e: pop r15; pop rbp; ret;
0x00005555555594b7: pop rbp; jmp rax; nop word ptr [rax + rax]; pop rbp; ret;
0x00005555555562b1e: pop rbp; pop r12; cmove rax, rdx; ret;
0x0000555555555a8cb: pop rbp; pop r12; jmp 0x3860; nop dword ptr [rax + rax]; mov eax, 0xffffffff; ret;
0x0000555555555f240: pop rbp; pop r12; jmp 0x5ed0; nop dword ptr [rax + rax]; mov eax, 0xffffffff; ret;
0x0000555555558b12: 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?



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: irc.freenode.net ##gef
- 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