Debugging Tools Intro

DWARF, ELF, GDB/binutils, build-id

Red Hat

Author Jan Kratochvíl <jan.kratochvil@redhat.com>

February 12, 2011

Agenda

1. Memory debugging tools
2. DWARF debug info
3. Unwinding using .eh_frame
4. Unique binaries identification by build-id

Section 1

Memory debugging tools

redhat.

Available memory debugging tools

- `valgrind [--db-attach=yes] executable params`
- `gcc -fstack-protector -O`
  defaults: rpmbuild yes, gcc no
- `gcc -D_FORTIFY_SOURCE=2 -O`
  defaults: rpmbuild yes, gcc no
- `LD_PRELOAD=/usr/lib64/libefence.so` – ElectricFence
- `gcc -lmcheck` or `MALLOC_CHECK_=3 executable`
- `gcc -fmudflap -lmudflap`
- `gdb --args executable params`

redhat

valgrind –db-attach=yes

- --suppressions=/usr/share/doc/python-*/valgrind-python.supp
- --num-callers=50

```
  char *s = malloc (0x10);
  s[0x10] = 0;
==1735== Invalid write of size 1
==1735==    at 0x400522: main (valgrindtest.c:4)
==1735==  Address 0x4c31050 is 0 bytes after a block of size 16 al
==1735==    at 0x4A05E46: malloc (vg_replace_malloc.c:195)
==1735==    by 0x400515: main (valgrindtest.c:3)
==1735== ---- Attach to debugger ? --- [Return/N/n/Y/y/C/c] ---- y
==1735== starting debugger with cmd: /usr/bin/gdb -nw /proc/1954/f
[...]
0x0000000000400522 in main () at valgrindtest.c:4
4   s[0x10] = 0;
(gdb)
```

redhat.

gcc -fstack-protector -O

- defaults: rpmbuild yes, gcc no

```
void f (int i) {
  void *p = alloca (i);
  memset (p, 0, 0x50); }
int main (void) { f (1); return 0; }

*** stack smashing detected ***: ./stackprotectortest term
======= Backtrace: =========
/lib64/libc.so.6(__fortify_fail+0x37)[0x32a1cfe1b7]
/lib64/libc.so.6(__fortify_fail+0x0)[0x32a1cfe180]
./stackprotectortest[0x400590]
======= Memory map: ========
[...]
Aborted
```

redhat.

gcc -D_FORTIFY_SOURCE=2 -O

- defaults: rpmbuild yes, gcc no

```
void f (int x) {
  char s[2];
  memset (s, 0, x); }
int main (void) { f (3); return 0; }

*** buffer overflow detected ***: ./fortifytest terminated
======= Backtrace: =========
/lib64/libc.so.6(__fortify_fail+0x37)[0x32a1cfe1b7]
/lib64/libc.so.6[0x32a1cfc0e0]
./fortifytest[0x4004ff]
[...]
Aborted
```

redhat

LD_PRELOAD=/usr/lib64/libefence.so – ElectricFence

```
int main (void) {
  char *s = malloc (0x10);
  s[0x10] = 0;

(gdb) set env LD_PRELOAD=/usr/lib64/libefence.so
(gdb) run
Starting program: efencetest
  Electric Fence 2.2.2 Copyright (C) 1987-1999 Bruce Perens
Program received signal SIGSEGV, Segmentation fault.
0x00000000004004e2 in main () at efencetest.c:4
4   s[0x10] = 0;
(gdb) _
```

redhat.

gcc -lmcheck

- enabled system-wide by Fedora debugmode.rpm

```
char *s = malloc (0x10);
s[0x10] = 0;
free (s);
```

```
MALLOC_CHECK_=3 MALLOC_PERTURB_=85 ./executable
*** glibc detected *** ./mchecktest: free(): invalid pointe
======= Backtrace: =========
/lib64/libc.so.6(+0x773ba)[0x7f4e92aa73ba]
./mchecktest[0x400531]
[...]
Aborted
```

redhat.

gcc -fmudflap -lmudflap

```
void f (int x) {
  char *p = alloca (x);
  p[2] = 0; }
int main (void) { f (2); return 0; }

*******
mudflap violation 1 (check/write): time=1297291147.545274 ptr=0x155bb11 s
pc=0x7ffd94c8ab21 location=`mudflaptest.c:4:8 (f)'
      /usr/lib64/libmudflap.so.0(__mf_check+0x41) [0x7ffd94c8ab21]
      ./mudflaptest(f+0x8a) [0x40095e]
      ./mudflaptest(main+0xe) [0x40097e]
Nearby object 1: checked region begins 1B after and ends 1B after
mudflap object 0x155bb60: name=`alloca region'
bounds=[0x155bb10,0x155bb10] size=1 area=heap check=0r/0w liveness=0
alloc time=1297291147.545162 pc=0x7ffd94c89ef1

[...]
```

redhat.

gdb record testfile

```
void first (void (*secondptr) (void)) {
  (*secondptr) ();
}
int main (void) {
  first (NULL);
  return 0;
}
```

redhat.

gdb record

- .gdbinit: set record insn-number-max 200000

```
(gdb) record
(gdb) continue
Continuing.
0x0000000000000000 in ?? ()
(gdb) backtrace
#0  0x0000000000000000 in ?? ()
#1  0x000000000040049b in main () at jumpzero.c:6
(gdb) reverse-stepi
first (secondptr=0) at jumpzero.c:3
3   (*secondptr) ();
(gdb) backtrace
#0  first (secondptr=0) at jumpzero.c:3
#1  0x000000000040049b in main () at jumpzero.c:6
(gdb) _
```

redhat.

GDB CLI (command line interface)

- `$ info gdb`
  - Sample Session
- `(gdb) start` — stop at `main`
- `(gdb) step` — step into
- `(gdb) next` — step over
- `(gdb) print expression`
- `(gdb) continue` — continue execution
- `(gdb) break line number` — put breakpoint
- `(gdb) break function name` — put breakpoint
- `(gdb) watch variable name` — create watchpoint
- `(gdb) delete breakpoint number` — delete breakpoint
- `(gdb) quit`

redhat.

GDB front ends

- Eclipse CDT
- KDevelop
- Qt Creator
- Nemiver
- NetBeans
- GNU Emacs
- GDB TUI
- DDD
- Insight
- see `http://sourceware.org/gdb/wiki/GDB_Front_Ends`

Section 2

DWARF debug info

redhat.

ELF

- described by `/usr/include/elf.h`
- magic: `00000000  7F 45 4C 46 [...] .ELF[...]`
- overview: `readelf -a binary`, `objdump -x binary`
  - elfutils: `eu-readelf -a binary`
- generic ELF is gELF
  - arch ABIs documents: i386, x86_64, ia64 etc.
- one of its debug info formats: DWARF

redhat.

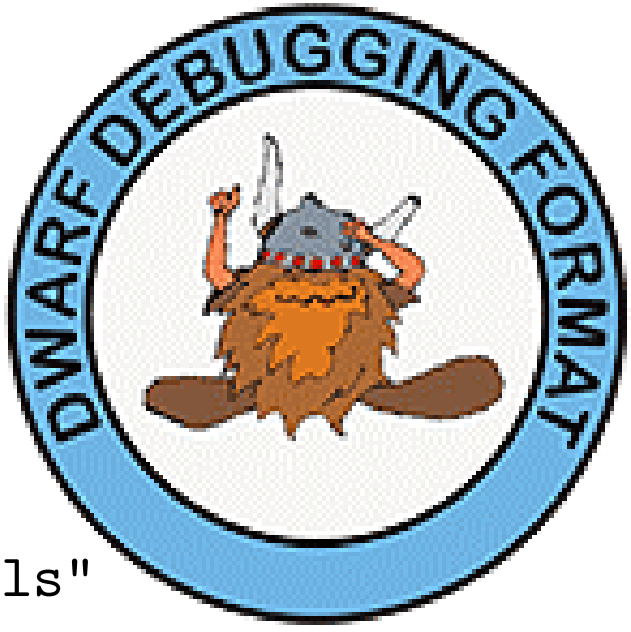
ELF sample

`readelf -a binary`

```
ELF Header:
  Class:                             ELF64
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Entry point address:               0x41aef0
Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg
  [ 1] .interp           PROGBITS        0000000000400238 000238 00001c 00   A
  [13] .plt              PROGBITS        000000000041a280 01a280 000c70 10  AX
  [14] .text             PROGBITS        000000000041aef0 01aef0 086268 00  AX
  [29] .debug_info       PROGBITS        0000000000000000 00c63f 088c33 00
Dynamic section at offset 0xd46d8 contains 26 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED) Shared library: [libtinfo.so.5]
 0x0000000000000001 (NEEDED) Shared library: [libc.so.6]
```

redhat.

How the DWARF looks




```
subprogram
name         (strp) "have_minimal_symbols"
decl_file    (data1) 1
decl_line    (data2) 997
type         (ref4) [ 11917]
low_pc       (addr) 0x486997 <have_minimal_symbols>
high_pc      (addr) 0x4869da <qsort_cmp>
frame_base (block1) [   0] call_frame_cfa
sibling      (ref4) [ 1ada7]
 variable

[...]
```

redhat.

DWARF

- specification: `http://dwarfstd.org`
- displayed by `readelf -w binary`
- ELF sections `.debug_*` (like `.debug_info`)
- DWARF versions in use are 2, 3 and 4
- gcc debug info level 3 provides macro information
- `gcc -g2` or `-g3` specify debug info level, not DWARF version
  - rpmbuild uses `-g`, that is like `-g2` (level 2)

redhat.

File formats in use

OS	file format	debug info format
GNU/Linux	ELF	DWARF
GNU/Linux	ELF	STABS
Apple OSX	Mach-O	DWARF
MS-Windows	PE32	PDB
MinGW32	PE32	DWARF

redhat.

STABS

- obsolete debug info format
- ELF sections `.stab`, `.stabstr`
- `gcc -gstabs+`

redhat.

DWARF parsing

- `elfutils-libs`
- `libdwarf`
- `gdb dwarf2read.c`
- `readelf -w` / `eu-readelf -w`

redhat.

gcc, gcc -s and gcc -g

- `.dynsym` is always present for shared libraries
  - `gcc -rdynamic` forces `.dynsym`
- `.symtab` is generated by default – for linkage
- `.debug_*` is generated by `gcc -g`
- `gcc -s` is like `gcc` + `strip` (no `.symtab`, no `.debug_*`)
  - `strip` by default removes both `.symtab` and `.debug_*`
- Both `.symtab` and `.debug_*` are in `*-debuginfo-*.rpm`
- `.symtab` (ELF) is used during linking

```
Symbol table '.symtab' contains 8602 entries:
Num:    Value  Size Type    Bind   Vis      Ndx Name
 76: 00016c93   124 FUNC    LOCAL  DEFAULT   12 init
```

redhat.

no runtime overhead

- (only `.dynsym` has runtime overhead)
- neither separate (`/usr/lib/debug/`) nor in-file debug info
- debug info is never mapped to memory
- debug info sections are not covered by segments at all
- ELF sections are for linking/debugging:

```
Section Headers:
[Nr] Name              Type     Addr     Off    Size   ES Flg
[13] .text             PROGBITS 0805e100 016100 0871dc 00  AX
[29] .debug_info       PROGBITS 00000000 00c63f 088c33 00
```

- ELF segments are mapped for runtime:

```
Program Headers:
Type Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg
LOAD 0x000000 0x08048000 0x08048000 0xcecc4 0xcecc4 R E
```

redhat.

DWARF sections

- `.debug_info`: `readelf -wi`: DIEs
  - `.debug_loc`: `readelf -wo`: associated `gcc -O2` ranges
  - `.debug_ranges`: `readelf -wR`: discontiguous functions
- `.debug_line`: `readelf -wl`: source lines $\leftrightarrow$ PC addresses
- `.debug_frame`: `readelf -wf`: unwinding
  - usually present as `.eh_frame`

Section 3

Unwinding using .eh_frame

redhat.

-fno-omit-frame-pointer stack layout

`-fno-omit-frame` easily unwinds but it steals %rbp (%ebp):

```
400474: 55          push   %rbp
400475: 48 89 e5    mov    %rsp,%rbp
[function body]
4004c3: c9          leaveq
4004c4: c3          retq
```

stack (high–to–low addresses, callers–to-callees, outer–to-inner):

- 0x7fffffffdc38 . . .
- 0x7fffffffdc30 frame #1 return address
- 0x7fffffffdc28 frame #1 saved `%rbp`
- 0x7fffffffdb50 frame #1 local variables. . .
- 0x7fffffffdb48 frame #0 return address
- 0x7fffffffdb40 frame #0 saved `%rbp` — `%rbp` points here
- 0x7fffffffdb28 frame #0 local variables. . . — `%rsp` points here

redhat.

.eh_frame benefits

- i686 default: `-fno-omit-frame-pointer` (<=gcc–4.5)
- x86_64 default: `-fomit-frame-pointer`
- `-fno-omit-frame-pointer` = 4% SPECint2000 i7 perf. hit
- `-fno-omit-frame` easily unwinds but it steals %rbp (%ebp):

```
400474: 55          push   %rbp
400475: 48 89 e5    mov    %rsp,%rbp
[function body]
4004c3: c9          leaveq
4004c4: c3          retq
```

- `-fomit-frame-pointer` unwinds using .eh_frame:

```
[function body]
4004bf: c3          retq
```

redhat.

.eh_frame

- used for runtime exceptions (covered by segments)
  - so-called unwinders
- not a part of DWARF (GNU extension)
  - it corresponds to the DWARF section `.debug_frame`
- no overhead when no exception is thrown
- `.eh_frame_hdr` is its runtime acceleration index
- rpmbuild default: `-fasynchronous-unwind-tables`
- used by `backtrace()`, `libunwind`, `gdb`, `SystemTap`

redhat.

.eh_frame sample code

```
0000000000000000 <functionname>:
   0:   48 83 ec 18                sub  $24,%rsp
   4:   c7 44 24 0c 00 00 00 00    movl $0,0xc(%rsp)

00000000 00000014 00000000 CIE
  DW_CFA_def_cfa: r7 (rsp) ofs 8
  DW_CFA_offset: r16 (rip) at cfa-8
00000018 00000014 0000001c FDE cie=00000000 pc=00000000..00
  DW_CFA_advance_loc: 4 to 00000004
  DW_CFA_def_cfa_offset: 32
```

redhat.

Section 4

Unique binaries identification by build-id

redhat.

build-id

- uniquely generated for each linked executable / shared library
- `eu-readelf -n file`

```
    Owner          Data size  Type
    GNU                   20  GNU_BUILD_ID
      Build ID: d48a....c8d1
```

- `/usr/lib/debug/.build-id/d4/8a....c8d1`
  → `../../../../../bin/bash` = `/bin/bash`
  `/usr/lib/debug/.build-id/d4/8a....c8d1.debug`
  → `../../bin/bash.debug` = `/usr/lib/debug/bin/bash.debug`
  - both symlinks are only in *-debuginfo-*.rpm
- list of build-ids from a core file: `eu-unstrip -n corefile`

The end.

Thanks for listening.