调试

约瑟夫·哈利特

2023 年 1 月 18 日

这到底是怎么回事？

写程序很难▶当出现问题
时我们应该有策略和工具
让我们向您指出一些！

示例程序

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) {
    字符消息[128]； size_t
    消息长度 = 256;字符时间戳[128]；
    时间_t t;结构 tm *tmp;


    FILE *file = fopen(argv[1],    a+    );

    printf(    请输入您的日志：     );
    getline(&message, &message_len, stdin);

    t = 时间(NULL);
    tmp = 当地时间(&t);
    strftime(时间戳, 256,     %C    , tmp);

    fprintf(文件,    %s: %s\n    , 时间戳, 消息);返回0；

}
```

让我们编译吧！

制作日记

ccjournal.c -ojournaljournal.c:在函数“main”中:journal.c:14:11:警告:从不兼容的指针类型传递“getline”的参数1[-Win兼容指针类型]

在 Journal.c:1 包含的文件中：/usr/include/stdio.h:645:45:注意:预期为“char * limit”,但参数的类型为“char ()[128]”

当我们跑步时...

./journal<<<“世界你好！”
分段错误（核心转储）

好的,让我们尝试调试一下

```
        gdb ./journal
从 ./journal 读取符号...
（在./journal中找不到调试符号）(gdb)运行<<<“hello”

启动程序：/home/joseph/Repos/Talks/COMS10012-Software-Tools/Debugging/journal <<<  hello
[启用使用libthread_db进行线程调试]
使用主机 libthread_db 库“/lib64/libthread_db.so.1”。

程序收到信号 SIGSEGV,分段错误。 __vfprintf_internal (s=0x0,
format=0x402026    %s: %s\n    , ap=ap@entry=0x7fffffffde50, mode_flags=mode_flags@entry=0) 在 vfpr 722^^I 东方； (gdb) bt #0 __vfprintf_internal
(s=0x0, 格式
=0x402026
  %s: %s\n    ,
      ap=ap@entry=0x7fffffffde50, mode_flags=mode_flags@entry=0) 在
      vfprintf-internal.c:722 #1
0x00007ffff7e2360a 在 __fprintf (stream=<优化输出>, format=<优化输出>) 在
      fprintf.c:32 #2 main()中的0x0000000000040125f
```

让我们变得更容易一点

```
    cc -Og -g 日志.c -o 日志 gdb ./journal
    (gdb) run
    <<<    hello
启动程序：/home/joseph/Repos/Talks/COMS10012-Software-Tools/Debugging/journal <<<    hello
[启用使用libthread_db进行线程调试]
使用主机 libthread_db 库“/lib64/libthread_db.so.1”。

程序收到信号 SIGSEGV,分段错误。 __memcpy_avx_unaligned_erms
() 位于 ../sysdeps/x86_64/multiarch/memmove-vec-unaligned-erms.S:333 下载 0.01 MB 源文件 /usr/src/debug/
glibc-2.36.9000-19.fc38.x86_64/string/ ../sysdeps/x86_64/multiarch/memmove-v 333^^I^^Imovl^^I%ecx, -4(%rdi, %rdx) (gb) bt #0
__memcpy_avx_unaligned_erms ()


        在 ../sysdeps/x86_64/multiarch/memmove-vec-unaligned-erms.S:333
#1 0x00007ffff7e496ac 在 __GI___getdelim (
        lineptr=lineptr@entry=0x7fffffffdff0、n=n@entry=0x7fffffffdfe8、
        delimiter=delimiter@entry=10、fp=0x7ffff7fa5aa0 <_IO_2_1_stdin_>) at
        iogetdelim.c:111 #2
0x00007ffff7e237d1 in __getline (lineptr=linept) r@entry=0x7fffffffdff0 ,
        n=n@entry=0x7fffffffdfe8, stream=<优化输出>) 在 getline.c:28 #3
0x00000000004011d6 in main (argc=<优化输出>, argv=<优化输出>) 在journal.c:14
```

看起来journal.c第14行一切都出了问题.....

(gdb) bjournal.c:14
0x4011ba 处的断点 2:文件journal.c,第 14 行。 (gdb) run
<<< hello
正在调试的程序已经启动。
从头开始吧？ （y 或 n)y 启动程序：/home/joseph/
Repos/Talks/COMS10012-Software-Tools/Debugging/journal <<< hello
[启用使用libthread_db进行线程调试]
使用主机 libthread_db 库“/lib64/libthread_db.so.1”。

断点 2,main (argc=<optimized out>, argv=<optimized out>) 位于 Journal.c:14 14^^I
getline(&message, &message_len, stdin); (gdb) 检查消
息 $3 =
@\000\000\000\000\000\000\000\000\200 , \000 <重复 14 次>, \006\000\000\000\216 \000\000\000\f\000\000\000\b (gdb) inform
message_len $4 = 256 (gdb) d
删除所有有
断点？（y
或 n） y (gdb)

如果有疑问……请阅读手册

在 man 3 getline

中： getline() 从流中读取整行，并将包含文本的缓冲区的地址存储到 *lineptr 中。该缓冲区以空字符结尾，并且包含换行符（如果找到）。

如果 *lineptr 在调用之前设置为 NULL，则 getline() 将分配一个缓冲区来存储该行。即使 getline() 失败，用户程序也应释放该缓冲区。
或者，在调用 getline() 之前，*lineptr 可以包含指向 malloc(3) 分配的缓冲区 *n 字节大小的指针。如果缓冲区不够大，无法容纳该行，则 getline() 使用 realloc(3) 调整其大小，并根据需要更新 *lineptr 和 *n。

好吧，我们正在传递一个静态分配的缓冲区……让我们解决这个问题。

一个新的*示例程序

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) { char
    *message = NULL; size_t 消
    息长度;字符时间戳[128];
    时间_t t;结构 tm *tmp;


    FILE *file = fopen(argv[1],    a+    );

    printf(    请输入您的日志：     );
    getline(&message, &message_len, stdin);

    t = 时间(NULL); tmp
    = 当地时间(&t); strftime(时
    间戳, 256,    %C    , tmp);

    fprintf(文件,    %s: %s\n    , 时间戳, 消息);返回0;

}

cc -g -Og 日记2.c -o 日记2
```

现在当我们跑步时……

$ ./journal2 <<< “你好”
分段错误（核心转储）

gdb ./journal2
(gdb) 运行<<< “你好”
启动程序：/home/joseph/Repos/Talks/COMS10012-Software-Tools/Debugging/journal2 <<< hel

程序收到信号 SIGSEGV,分段错误。 0x00007ffff7e2de82 in
__vfprintf_internal () from /lib64/libc.so.6 缺少单独的调试信息,请使用:dnf
debuginfo-install glibc-2.36.9000-19.fc38.x86_64 (gdb) bt #0 0x00007ffff7e2de82 in __vfprintf_internal
() from /
lib6 4/ libc.so.6 #1 0x00007ffff7e2360a in fprintf () from /lib64/libc.so.6 #2
0x0000000000401225 in main (argc=<优化输出>, argv=<优化输出>) 在
journal2.c:20 (gdb)

……好吧,我们更进一步了……

我们可以继续使用 gdb

GDB 是一个极其强大的调试工具▶它也很难使
用▶明年参见计算机系统 B，
或硕士级别的系统和软件安全▶如果您使用的是 Mac 或 BSD 机器,请查看 lldb ▶或获取
适当的教程每次打开它时它都会向您推荐的文档。

非常值得您花时间学习……
▶但这课程是关于软件工具的,我想向您展示更多

斯特雷斯

strace 工具可让您跟踪程序使用的系统调用

- 在 OpenBSD 上,请参见 ktrace 和 kdump
- 在 MacOS/FreeBSD 上,请参见 dtruss 和 dtrace

让我们运行它吧！

$ strace ./journal2 <<< Hello execve(./
journal2 , [./journal2], 0x7ffd3ef71360 /* 36 vars */) = 0 brk(NULL) = 0x154f000 arch_prctl(0x3001 /*
ARCH_? ?? */, 0x7ffc01b1b610) = -1 EINVAL (无效参数) access(/etc/
ld.so.preload , R_OK)
= -1 ENOENT（没有这样的文件或目录） openat(AT_FDCWD, /
etc/ld.so.cache , O_RDONLY|O_CLOEXEC) = 3 newfstatat(3, , {st_mode=S_IFREG|0644,
st_size=74509, ...}, AT_EMPTY_PATH) = 0 mmap(NULL, 74509, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff1ca8ef000 close(3)
openat(AT_FDCWD, /lib64/libc.so.6 , O_RDONLY|O_CLOEXEC) = 3读(3,
= 0
\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P|\2\ 0\0\0\0\0 ..., 832)... pread64(3,
\6\0\0\0\4\0\0\0@\0\0\0\0\ 0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0 ..., newfstatat(3, , {st_mode= S_IFREG|0755,
st_size=2232840, ...}, AT_EMPTY_PATH) ... mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0) = 0x7f... pread64(3, \6\ 0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0 \0\0\0\0 ..., mmap(NULL, 1961264,
PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff1ca70e00... mmap(0x7ff1ca734000, 1409024, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DE. .. mmap(0x7ff1ca88c000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3,... mmap(0x7ff1ca8df000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DEN...
mmap(0x7ff1ca8e5000, 32048, PROT_READ|PROT_WRITE, MAP_PRIVATE |MAP_FIXED|MAP_ANO... 关闭(3) = 0 mmap(NULL, 8192,
PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f... arch_prctl(ARCH_SET_FS, 0x7ff1ca8ee640) = 0
set_tid_address(0x7ff1ca8ee910) =第34069章 set_robust_list(0x7ff1ca8ee920, 24) = 0 rseq(0x7ff1ca8eef60, 0x20, 0, 0x53053053) = 0
mprotect(0x7ff1ca8df000, 16384, PROT_READ) = 0 mprotect(0x403000, 4096, PROT _READ) mprotect(0x7ff1ca933000, 8192,
PROT_READ) = 0 prlimit64 (0, RLIMIT_STACK, NULL,
{rlim_cur=9788*1024, rlim_max=RLIM64_INFINITY}... munmap(0x7ff1ca8ef000, 74509) = 0
getrandom(\x9e\xe1\xb9\x13\x31\x2c\x9e\xee , 8、
GRND_NONBLOCK) = 8 brk(NULL) = 0x154f000 brk(0x1570000) = 0x1570000

= 0

openat(AT_FDCWD, NULL, O_RDWR|O_CREAT|O_APPEND, 0666) = -1 EFAULT (错误地址) newfstatat(1, , {st_mode=S_IFCHR|
0620, st_rdev=makedev(0x88, 0x5), ...}, AT_EMPTY_PAT newfstatat(0, , {st_mode=S_IFIFO|0600, st_size=0, ...}, AT_EMPTY_PATH) = 0
读取(0, hello\n , 4096) = 6 openat(AT_FDCWD, /etc/本地时间 , O_RDONLY|O_CLOEXEC) = 3 newfstatat(3, ,
{st_mode=S_IFREG|0644, st_size=3664, ...}, AT_EMPTY_PATH) = 0
newfstatat(3, , {st_mode=S_IFREG|0644, st_size=3664, ...}, AT_EMPTY_PATH) = 0 读取(3,
TZif2\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\ 0\0\0\10\0\0\0\10\0\0\0\0 ..., 4096) = 366 lseek(3, -2329, SEEK_CUR) 读取(3,
TZif2\0\ 0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\10\0\0\0\10\0\0\0\ 0 ..., 4096) = 232 关闭(3)

= 1335

= 0

--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0xc0} --- 被 SIGSEGV 杀死（核心转
储） 分段错误（核心转储）

输出太多了！

strace 允许您使用正则表达式来过滤您查看的系统调用

- ...或者你可以只使用 grep... $ strace

-e /open.* ./journal2 <<<hello openat(AT_FDCWD, /etc/ld.so.cache , O_RDONLY|O_CLOEXEC) = 3 openat(AT_FDCWD , / lib64/libc.so.6 , O_RDONLY|O_CLOEXEC) = 3 openat(AT_FDCWD, NULL, O_RDWR|O_CREAT|O_APPEND, 0666) = -1 EFAULT (错误地址) openat(AT_FDCWD, /etc/localtime , O_RDONLY|O_CLOEXEC) = 3 --- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0xc0} --- 被 SIGSEGV 杀死（核心转储） 分段错误（核心转储）

哦,是的……我们忘记了一个参数

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) { char
    *message = NULL; size_t 消
    息长度;字符时间戳[128];
    时间_t t;结构 tm *tmp;


    FILE *file = fopen(argv[1],    a+    ); /* 第 11 行 */

    printf(    请输入您的日志：     );
    getline(&message, &message_len, stdin);

    t = 时间(NULL); tmp
    = 当地时间(&t); strftime(时
    间戳, 256,    %C    , tmp);

    fprintf(文件,    %s: %s\n    , 时间戳, 消息); /* 第 20 行 */ 返回 0;

}
```

让我们解决这个问题...

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) { char
    *message = NULL; size_t 消
    息长度;字符时间戳[128];
    时间_t t;结构 tm *tmp;



    if (argc < 2) { printf(    使用 %s 路径/to/log\n    , argv[0]);退出（1）；  };
    FILE *file = fopen(argv[1],    a+    ); /* 第 11 行 */

    printf(    请输入您的日志：     );
    getline(&message, &message_len, stdin);

    t = 时间(NULL); tmp
    = 当地时间(&t); strftime(时
    间戳, 256,    %C    , tmp);

    fprintf(文件,    %s: %s\n    , 时间戳, 消息); /* 第 20 行 */ 返回 0; }
```

现在当我们跑步时！

./journal3 文档/log.txt <<<hello 分段错误（核心转储）

这次让我们尝试一下 ltrace（其他平台上没有类似的功能）...

- 它跟踪库调用

ltrace 和更多 strace

```
$ ltrace ./journal3 文档/log.txt <<<hello fopen(    documents/
log.txt    ,    a+    ) printf(    输入日志：    )                                                    = 零 =
getline(0x7fffebcc0fc8,                                                                              15 =
0x7fffebcc0fc0, 0x7f4bfcf40aa0, 0) time(nil ) localtime(0x7fffebcc0f38)                               6 =
                                                                                                     1674045150 =
strftime(    20    , 256,    %C    ,                                                                 0x7f4bfcf47640 = 2
0x7f4bfcf47640) fprintf(nil,    %s: %s\n    ,    20    ,
   hello\n    <无返回...> --- SIGSEGV（分段错误） ---            被 SIGSEGV 杀
死


$ strace -e openat ./journal3 文档/log.txt <<<hello openat(AT_FDCWD,
   /etc/ld.so.cache    , O_RDONLY|O_CLOEXEC) = 3 openat(AT_FDCWD,    /
lib64/libc.so. 6    , O_RDONLY|O_CLOEXEC) = 3 openat(AT_FDCWD,
   documents/log.txt    , O_RDWR|O_CREAT|O_APPEND, 0666) = -1 ENOENT (没有这样的文件或目录) openat(AT_FDCWD,    /etc/localtime    ,
O_RDONLY|O_CLOEXEC) = 3 --- SIGSEGV {si_signo=SIGSEGV,
si_code=SEGV_MAPERR, si_addr=0xc0} ---            被 SIGSEGV 杀死（核心转储）            分
段错误（核心转储）
```

让我们解决这个问题...

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <errno.h>

int main(int argc, char *argv[]) { char
    *message = NULL; size_t 消
    息长度;字符时间戳[128];
    时间_t t;结构 tm *tmp;



    if (argc < 2) { printf(    使用 %s 路径/to/log\n    , argv[0]);退出（1）；  };
    FILE *file = fopen(argv[1],    a+    ); /* 第 11 行 */ if (file
    NULL) { perror(    无法打
        开日志    );退出（2）；  }



    printf(    请输入您的日志：      );
    getline(&message, &message_len, stdin);

    t = 时间(NULL); tmp
    = 当地时间(&t); strftime(时
    间戳, 256,    %C    , tmp);

    fprintf(文件,    %s: %s\n    , 时间戳, 消息); /* 第 20 行 */ 返回 0;

}
```

现在当我们跑步时...

```
$ ./journal4 <<<hello 用法 ./
journal4 路径/到/日志

$ ./journal4 文件/log.txt <<<hello 无法打开日志 :没有这样的文
件或目录

$ ./journal4 /etc/passwd <<<hello 无法打开日志 :权
限被拒绝

$ ./journal4 /dev/stdout 输入日志：
hello 20: hello
```

20？！

来自 man 3 strftime：

%c 当前语言环境的首选日期和时间表示形式。 （当前语言环境中使用的特定格式可以通过调用 nllanginfo(3)来获取,其中DTFMT作为 %c 转换规范的参数， ERADTFMT作为 %Ec 转换规范的参数。）（在 POSIX 语言环境中,这是相当于 %a %b %e %H:%M:%S %Y。）

%C 世纪数（年/100）,为 2 位整数。 (SU)（%EC 转换规范对应于时代名称。）（从tmyear 计算。）

调试工具无法发现写得不好的代码！

但其他工具可以捕获一些东西.....

回想一下当我们修复 getline 时……它说它将为该行分配内存

- ……我们释放过它吗？

$ valgrind ./journal4 /dev/stdout <<<hello 36111

Memcheck,内存错误检测器 36111 版权所有 (C) 2002-2022，

GNU GPL,作者:Julian Seward 等人。 36111 使用Valgrind-3.20.0和LibVEX;使用 -h 重新运行以获取版权信息

36111 命令：./journal4 /dev/stdout 36111 20: hello

输入日志： 36111

36111 堆摘要:退出时使用:2

情况:13 次分 个块中的 592 字节 36111 36111 总堆使用

配、11 次释放、已分配 13,684 字节 36111 36111 泄漏摘要:肯定丢失:1个块中的120个字节 36111 36111

间接丢失:0块中

的0个字节 361111可能丢失了:0 0

36111 个块 36111 仍然可达:1 个块中的 472 个字节

已抑制:0 个块中的 0 个字节

包起来

在本次讲座中,我们重点介绍了几种调试工具的基础知识

- strace、ltrace、valgrind 和 gdb 将帮助处理您遇到的大多数错误

但良好的防御性编程策略也是如此▶始终检查函数的返回码▶始终检查假设▶始终修复编译器警告

…实际上收到更多警告！

使用 -Wall -Wextra　　std=c11 -pedantic 进行编译将使编译器对你的 C 代码非常挑剔……

但是还有其他称为linter的工具可以变得更加挑剔C/C++ Clang Static Analyser、Rats Java FindBugs

Haskell hlint

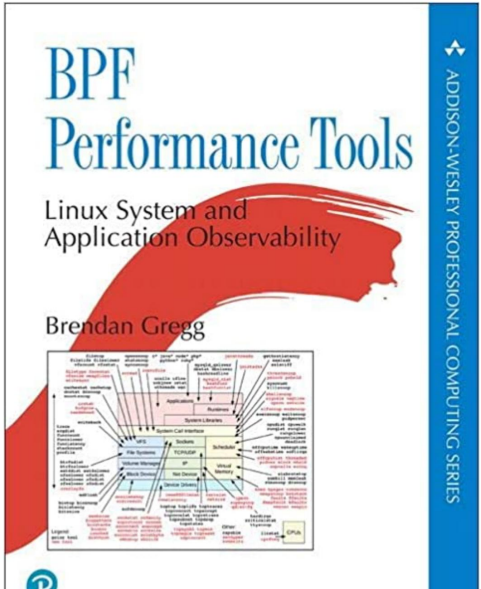
Python pylint,mypy

其他 C/C++ 工具可以添加额外的运行时检查

ASan地址消毒剂;检查指针恶作剧

UBSan未定义行为消毒剂;检查 C 陷阱

带通滤波器工具



Linux 有一个（合理的)新的仪器框架,称为eBPF

▶它可以让您获得有关程序正在执行的操作的大量详细信息▶高度 Linux 特定性▶

我需要学习它 :-(