

理解程序运行

使用

本文档以通关方式撰写，完成一关进入下一关，请将需要填写的内容写在空白处。

概述

这个文档用来帮助大家理解程序运行原理。

GATE 1

基础环境

请安装 32 位 CentOS 操作系统，安装版本 6.6

安装后以 root 用户执行：

```
yum install gcc  
yum install gdb
```

静态二进制分析

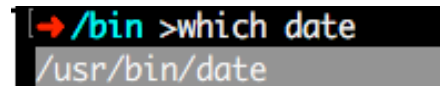
使用 linux 命令 **date**. 启动控制台窗口，输入如下命令。

```
date  
date --date="4 hours ago"  
date --date="2 years ago" +%Y%m%d
```

date 程序在 Linux 操作系统的什么位置？你用什么指令找到的该位置？

date 位于 linux 操作系统的/bin 中，具体为/usr/bin/date

使用 which date 命令查找，如下图：



```
➔ /bin >which date  
/usr/bin/date
```

简要介绍一下 **objdump** 的功能.

通过 man objdump 可以看到其主要功能是 display information from object files，查看二进制文件并可以进行反汇编等操作。

在控制台输入如下指令，用找到的地址替换指令中的{loc}，**简要解释**你看到的内容。

```
objdump -d {loc}/date | less
```

```
>objdump -d /bin/date|less -N
```

其是对 **date** 文件的反汇编并使用 **less** 工具对反汇编的内容进行带行号的查看，反汇编内容如下：

```

1
2 /bin/date:      文件格式 elf64-x86-64
3
4
5 Disassembly of section .init:
6
7 0000000000401620 <.init>:
8 401620: 48 83 ec 08      sub    $0x8,%rsp
9 401624: 48 8b 05 ad c9 20 00 mov    0x20c9ad(%rip),%rax      # 60dfd8 <__sprintf_chk@plt+0x20c528>
10 >
11 40162b: 48 85 c0         test   %rax,%rax
12 40162e: 74 05           je     401635 <__ctype_toupper_loc@plt-0x1b>
13 401630: e8 7b 02 00 00   callq 4018b0 <__gmon_start__@plt>
14 401635: 48 83 c4 08      add    $0x8,%rsp
15 401639: c3             retq
16
17 Disassembly of section .plt:
18
19 0000000000401640 <__ctype_toupper_loc@plt-0x10>:
20 401640: ff 35 c2 c9 20 00 pushq  0x20c9c2(%rip)      # 60e008 <__sprintf_chk@plt+0x20c558>
21 401646: ff 25 c4 c9 20 00 jmpq    *0x20c9c4(%rip)      # 60e010 <__sprintf_chk@plt+0x20c560>
22 40164c: 0f 1f 40 00      nopl   0x0(%rax)
23
24 0000000000401650 <__ctype_toupper_loc@plt>:
25 401650: ff 25 c2 c9 20 00 jmpq    *0x20c9c2(%rip)      # 60e018 <__sprintf_chk@plt+0x20c568>
26 401656: 68 00 00 00 00   pushq  $0x0
27 40165b: e9 e0 ff ff ff   jmpq    401640 <__ctype_toupper_loc@plt-0x10>
28
29 0000000000401660 <__uflow@plt>:
30 401660: ff 25 ba c9 20 00 jmpq    *0x20c9ba(%rip)      # 60e020 <__sprintf_chk@plt+0x20c570>
31 401666: 68 01 00 00 00   pushq  $0x1
32 40166b: e9 d0 ff ff ff   jmpq    401640 <__ctype_toupper_loc@plt-0x10>
33
34 0000000000401670 <getenv@plt>:
35 401670: ff 25 b2 c9 20 00 jmpq    *0x20c9b2(%rip)      # 60e028 <__sprintf_chk@plt+0x20c578>
36 401676: 68 02 00 00 00   pushq  $0x2
37 40167b: e9 c0 ff ff ff   jmpq    401640 <__ctype_toupper_loc@plt-0x10>
38
39 0000000000401680 <free@plt>:
40 401680: ff 25 aa c9 20 00 jmpq    *0x20c9aa(%rip)      # 60e030 <__sprintf_chk@plt+0x20c580>
41 401686: 68 03 00 00 00   pushq  $0x3

```

GATE 2

输入如下指令，比较该指令与之前指令在输出上的区别，解释-xtrds 的功能。

```
objdump -xtrds {loc}/date | less
```

对程序内容进行分类显示，并带符号表，以程序头、动态节、版本引用、SYMBOL TABLE 等内容显示。

-xtrds 功能为：

x: 以某种分类信息的形式把目标文件的数据组成输出（Display all available header information, including the symbol table and relocation entries. Using -x is equivalent to specifying all of -a -f -h -p -r -t.）

t: 输出目标文件的符号表（Print the symbol table entries of the file. This is similar to the information provided by the nm program, although the display format is different. The format of the output depends upon the format of the file being dumped, but there are two main types. ）

r: 输出文件相关的入口（Print the relocation entries of the file. If used with -d or -D, the relocations are printed interspersed with the disassembly.）

d:对文件进行反汇编（Like -d, but disassemble the contents of all sections, not just those expected to contain instructions.）

s: 输出文件各个节的全部内容。（Display the full contents of any sections requested. By default all non-empty sections are displayed.）

综上，-xtrds 功能为对某个二进制文件进行反汇编，并对文件及文件相关入口的各个节的内容进行带符号表的显示。

在控制台输入如下命令，将输出复制到空白处。

```
objdump -xtrds {loc}/date | grep bugs
```

```
40bb60 74206275 67732074 6f3a2025 730a0062 t bugs to: %s..b
```

GATE 3

将 **date** 文件拷贝到你自己的目录中，用 **vim** 打开文件（也可以用其他工具），输入如下命令：

```
:%s/bugs/dogs/g
```

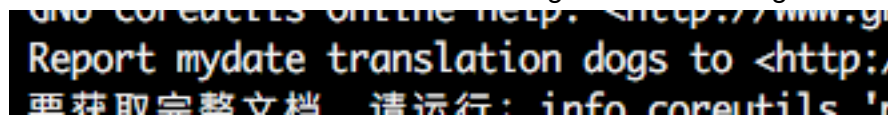
 #注：进入 vim 后，使用 **shift+ :**，然后输入本行内容（首个：不需要输入）

该命令将 **date** 文件中的 **bugs** 字符串替换成 **dogs** 字符串，保存后退出（**shift + :** 然后输入 **wq**），然后输入如下命令

```
./date -help
```

解释你看到的输出。

直接对源二进制文件进行了修改，并将 **bugs** 全部替换为 **dogs**，效果如下：

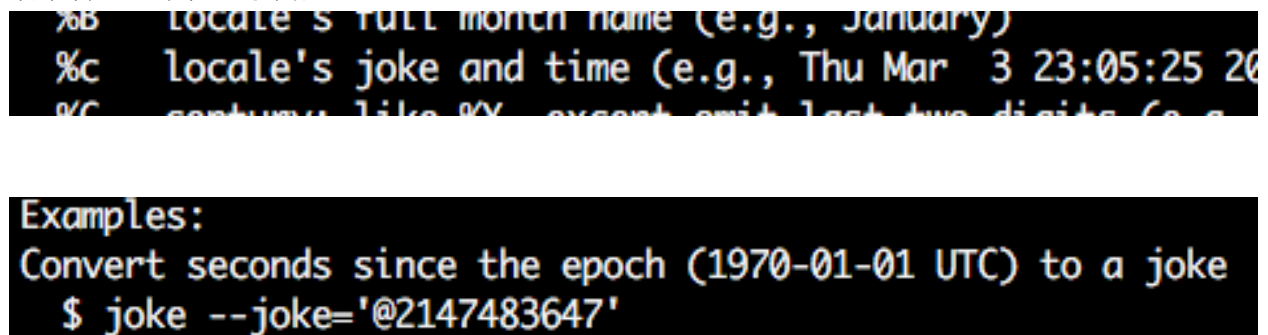


```
Report mydate translation dogs to <http://www.gnu.org/bugs/translation>
要获取完整文档，请运行：info coreutils 'date'
```

课后熟悉使用 **vim** 工具（或其他编辑工具，建议使用十六进制编辑器）对该程序进行其他更有意思的修改，请将课后完成的内容填写在下面的空白处。（此处为必须完成的内容）

```
:%s/date/joke/g
```

效果明显，下任意取两处：



```
%B locale's full month name (e.g., January)
%c locale's joke and time (e.g., Thu Mar 3 23:05:25 2000)
%C century, like %Y, except with last two digits (e.g., 20)

Examples:
Convert seconds since the epoch (1970-01-01 UTC) to a joke
$ joke --joke='@2147483647'
```

GATE 4

另外一个有用工具是 **readelf**. ELF 是 Linux 系统的目标文件格式，相当于 windows 系统中的 PE 格式。依次执行如下命令，在空白处依次解释各命令的基本功能：

```
readelf -l {loc}/date  
readelf -S {loc}/date  
readelf -W -s {loc}/date  
readelf -x 16 {loc}/date
```

- 1、以段形式显示程序头文件
- 2、显示程序的部分头文件
- 3、显示 **elf** 的符号表
- 4、以十六进制的形式显示某个 **section**

GATE 5

使用 gdb

在命令行输入如下命令：

```
gdb
```

```
(gdb) help
```

利用这个 **help** 命令了解 **gdb** 的使用，例如：**help running**。

请把你做过的练习（只保留命令）填写在空白处。

(必做内容)

```
gdb -x tools/gdbinit #使用配置文件对进行调试
```

```
(gdb) si #单步调试
```

```
(gdb) b *0x7c00 #在0x7c00处设置一个断点
```

```
(gdb)c #继续执行
```

```
gdb date
```

```
(gdb) set args "--help"
```

```
(gdb) break __libc_start_main
```

```
(gdb) run
```

```
(gdb) frame
```

```
(gdb) bt
```

```
(gdb) info frame
```

```
(gdb) info registers
```

```
(gdb) x /16xw $esp
```

GATE 6

分析程序

在控制台输入如下命令：

```
gdb date
(gdb) run
(gdb) quit
```

再输入如下命令，体会该工具的作用：

```
gdb date
(gdb) set args "--help"
(gdb) break __libc_start_main
(gdb) run
(gdb) frame
(gdb) bt
(gdb) info frame
(gdb) info registers
(gdb) x /16xw $esp
```

探索更多寄存器内容。对于代码和堆栈位置，你有何发现？

调用函数时会建立新的栈
eip 指向当前（下一条）指令
esp 存储函数跳转前 eip 的内容

GATE 7

尝试如下内容：

```
gdb date
(gdb) set args "--help"
(gdb) maintenance info sections
记录.rodata 段的起始地址
(gdb) break __libc_start_main
(gdb) run
(gdb) x/20s {.rodata address}
记录字符串 "GNU coreutils"的真实地址, {start}
(gdb) set *{start} = 0x554c4200
(gdb) c
```

上述代码做了些什么？你能否做类似修改证明你理解上述指令含义？请将你的解释和可能的修改写在下面。

对"GNU coreutils"的内容进行修改

```
(gdb) set *0x40981f = 0x554c4200
```

GATE 8

下课了。