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 程序在/bin/date 中，通过 which 命令找到 which date

简要介绍一下 **objdump** 的功能.

对二进制文件进行反汇编，显示文件头、符号表、执行指令、Section Header 等信息

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 显示二进制文件可执行部分的汇编代码，后面接/bin/date 即显示 date 程序的代码区；less 是部分显示——整条语句输出了/bin/date 的代码区，包括了 ebp、esp 的入栈出栈以及其他指令调用等。

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

-d 仅输出代码区，-xtrds 输出代码区的同时，按特定格式输出了文件头信息、符号表、重定位条目、以及所选节点的详细信息：

-x, --all-headers Display the contents of all headers
-t, --syms Display the contents of the symbol table(s)
-r, --reloc Display the relocation entries in the file
-d, --disassemble Display assembler contents of executable sections
-s, --full-contents Display the full contents of all sections requested

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

8052a90 6174696f 6e206275 67732074 6f203c68 ation bugs to <h
80535a0 0a526570 6f727420 62756773 20746f3a .Report bugs to:

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。

在最后一段原本的" Report date bugs" 变成了 "Report date dogs"
原因：
Vim 命令中，s 为字符串替换，%为替换所有行中对应的字符串，/g 为替换此行中所有对应的字符串——故整条语句执行结果为：将所有的 bugs 替换为 dogs

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）

通过 IDA 将 date 反汇编，找到—version 模块的入口地址，在该入口地址处添加 shellcode，fork 出一个 shell。

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

-l --program-headers Display the program headers -S --section-headers Display the sections' header -w[lLiaprmfFsoR] Display the contents of DWARF2 debug sections -s --syms Display the symbol table -x --hex-dump=<number|name> Dump the contents of section <number|name> as bytes

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。

gdb ./date
b __libc_start_main
c
n
ni
s
si
bt

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？

1. 系统调用可能不支持任意大小的内存分配。有些系统的系统调用只支持固定大小及其倍数的内存请求(按页分配)；这样的话对于大量的小内存分类来说会造成浪费。
2. 系统调用申请内存可能是代价昂贵的。系统调用可能涉及用户态和核心态的转换。
3. 没有管理的内存分配在大量复杂内存的分配释放操作下很容易造成内存碎片。

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。

Gdb gate：gdb 打开二进制文件 gate
set args “--help”：设置 argv 参数
maintenance info sections：记录.rodata 段的起始地址
break __libc_start_main：设置断点
run：运行程序
x/20s {.rodata address}：记录字符串 “GNU coreutils”的真实地址, {start}
set *{start} = 0x554c4200：设置入口点
c：继续运行

可能的修改：break 断点可以修改到别处；args 也可以设置其他 argv 参数；记录地址亦可修改

GATE 8

下课了。