lab1 实验手册

网络与系统安全综合实验

实验截止: 2020.09.01 星期二 23:59

概要

欢迎来到网络与系统安全综合实验，在本课程中，我们将使用工业界主流的逆向工具 IDA 对 Linux 下的二进制程序（ELF 格式）进行逆向分析。同时根据逆向的结果找出程序中的漏洞，并且利用漏洞让程序到达非预期的运行结果。在本门课中，所有程序都基于 x86 架构在 Linux 下运行。我们强烈推荐使用虚拟机完成实验内容。

本指南为曲海鹏老师的《网络与系统安全综合实验》课程实验手册，有很多助教参与了实验的设计，编写等工作：

- 房建（2020）
- 吕文杰（2020）
- 张政（2020）

1 实验介绍

1.1 实验概述

本次课程一共有三个 lab：

- lab1：逆向实验
- lab2：shellcode& 栈溢出实验
- lab3：格式化字符串 &ROP 实验

每周布置一个 lab，每次都会有 4-5 个题目需要完成，每个 lab 实验时间为一周。实验以个人为单位。每次 lab 都会在截至日期后的下一节课检查。

1.2 提交内容

每次 lab 实验需要提交 pdf 格式实验报告，实验报告需要完整反映题目解答过程和最后的答案。如果编写了对应的解题脚本，连同脚本和实验报告一同压缩为.zip 格式上交。

2 IDA 使用方法

IDA 是一个在 Windows，Linux，MacOS 上的交互式，可编程，可拓展，多处理器的反汇编程序，用于将机器码转换成可读的汇编语言。本实验将使用 IDA 进行逆向分析，你可以从 `https://www.hex-rays.com/products/ida/support/download_freeware/` 获取免费版本的 IDA。

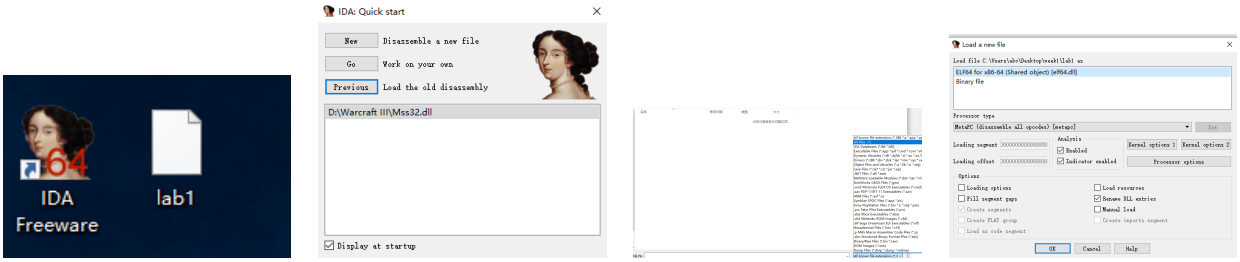
软件安装成功后，桌面上可以看到 IDA 的图标，如图 1（a）所示，双击点击 IDA 图标（也可直接将文件拖动到图标上）。

点击 new，选择文件如图 1（b）所示。

点击左下角选择 All Files，如图 1（c）所示。之后便可以看见所要逆向的文件

之后选择要逆向的文件点击打开。

选择文件的指令集如图 1（d）所示，一般情况下 IDA 已默认选择好对应的架构，选择好后点击 OK 后即可进入 IDA 的主界面，如图 2 所示。



(a) IDA 图标与待逆向文件　(b) IDA Quick start　(c) 文件选择　(d) 选择指令集

图 1: IDA 打开文件过程

在 IDA 中，右侧为 IDA 反汇编二进制文件产生的汇编代码。左侧为 IDA 扫描文件后识别出的函数，点击不同的函数名，右侧可显示出不同函数对应的汇编代码。左下角为当前函数对应的控制流图（CFG）。关于 IDA 其他更详细的操作，可查询 IDA 文档（`https://www.hex-rays.com/products/ida/support/`）

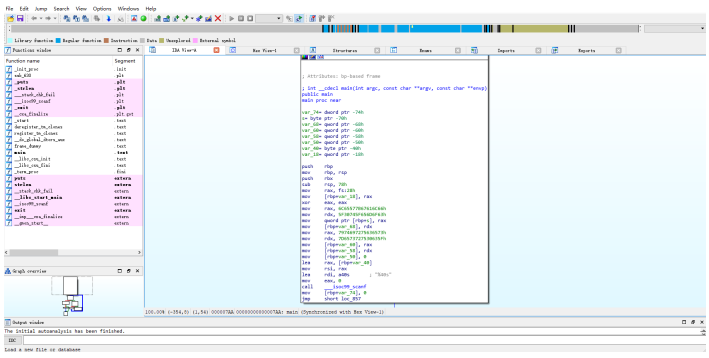


图 2: IDA 界面

3　gdb 使用方法

GNU 调试器（英语：GNU Debugger，缩写：GDB），是 GNU 软件系统中的标准调试器，此外 GDB 也是个具有移携性的调试器，经过移携需求的调修与重新编译，如今许多的类 UNIX 操作系统上都可以使用 GDB。

在本门课中，我们将使用 gdb（建议使用 7.12.1 及其以上版本）来动态调试二进制程序。一些 gdb 的插件会让这个过程更加方便，如 peda（`https://github.com/longld/peda`）等插件。常用的 gdb 命令如表 1 所示。其他 gdb 命令可查询 gdb 文档（`https://sourceware.org/gdb/onlinedocs/gdb/`）

表 1: gdb 常用命令

命令	参数	含义
run		运行当前程序
break	地址	在参数所表示的地址处添加断点
continue		从当前位置继续执行，直到遇到断点停止
quit		退出 gdb
attach	进程的 pid	使用 gdb 调试当前运行的程序
n		set over，遇到函数不进入函数内部
s		set into，遇到函数进入函数内部
info	registers	显示所有寄存器
x/nfu	地址	以 f 格式打印从地址处开始的 n 个长度单元为 u 的内存值。 f: 是输出格式。x:16 进制，o:8 进制。u: 标明一个单元的长度。 b: 一个 byte，h: 两个 byte（halfword） w: 四个 byte（word），g: 八个 byte（giant word）
vmmap(需要 peda 插件)		打印调试程序的虚拟地址映射

现在通过演示一个实验的例子来展示调试二进制程序的大体流程。

代码 1: 使用 gdb 打开要调试的文件

```
$ gdb ./lab1
GNU gdb (GDB) 7.12.1
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show␣copying"
and "show␣warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show␣configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos␣word" to search for commands related to "word"...

warning: ~/Pwngdb/pwngdb.py: No such file or directory
Reading symbols from ./lab1...(no debugging symbols found)...done.
gdb-peda$
```

在打开二进制程序之后，在 main 函数处下断点，之后运行文件

代码 2: gdb 下断点

```
gdb-peda$ b * main
Breakpoint 1 at 0x400657
gdb-peda$ r
Starting program: /mnt/hgfs/learn/security_course/week1/lab1
[----------------------------------registers-----------------------------------]
```

```
RAX: 0x400657 (<main>:  push   rbp)
RBX: 0x0
RCX: 0x400750 (<__libc_csu_init>:       push   r15)
RDX: 0x7fffffffe378 --> 0x7fffffffe62d ("LC_TERMINAL_VERSION=3.3.6")
RSI: 0x7fffffffe368 --> 0x7fffffffe602 ("/mnt/hgfs/learn/security_course/week1/lab1")
RDI: 0x1
RBP: 0x400750 (<__libc_csu_init>:       push   r15)
RSP: 0x7fffffffe288 --> 0x7ffff7a05b97 (<__libc_start_main+231>:        mov    edi,
    eax)
RIP: 0x400657 (<main>:  push   rbp)
R8 : 0x7ffff7dd0d80 --> 0x0
R9 : 0x7ffff7dd0d80 --> 0x0
R10: 0x0
R11: 0x0
R12: 0x400570 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffe360 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x400651 <frame_dummy+1>:    mov    rbp,rsp
   0x400654 <frame_dummy+4>:    pop    rbp
   0x400655 <frame_dummy+5>:    jmp    0x4005e0 <register_tm_clones>
=> 0x400657 <main>:     push   rbp
   0x400658 <main+1>:   mov    rbp,rsp
   0x40065b <main+4>:   push   rbx
   0x40065c <main+5>:   sub    rsp,0x78
   0x400660 <main+9>:   mov    rax,QWORD PTR fs:0x28
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe288 --> 0x7ffff7a05b97 (<__libc_start_main+231>:        mov    edi,
    eax)
0008| 0x7fffffffe290 --> 0x1
0016| 0x7fffffffe298 --> 0x7fffffffe368 --> 0x7fffffffe602 ("/mnt/hgfs/learn/
    security_course/week1/lab1")
0024| 0x7fffffffe2a0 --> 0x100008000
0032| 0x7fffffffe2a8 --> 0x400657 (<main>:     push   rbp)
0040| 0x7fffffffe2b0 --> 0x0
0048| 0x7fffffffe2b8 --> 0x50d55bbaf0a02160
0056| 0x7fffffffe2c0 --> 0x400570 (<_start>:   xor    ebp,ebp)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x0000000000400657 in main ()
gdb-peda$
```

使用 n 命令运行每一条汇编指令，可以看见每一条指令运行后寄存器和栈的变化。可以通过内存打印指令打印出

不同内存的内容。这里使用 x/30xg 打印出内存中对应地址中的值。

代码 3: gdb 打印内存

```
gdb-peda$ x/30xg 0x7fffffffe210
0x7fffffffe210: 0x6c65577b67616c66      0x5f30745f656d6f63
0x7fffffffe220: 0x7974697275636573      0x7d6573727530635f
0x7fffffffe230: 0x0000000000000000      0x0000000000f0b5ff
0x7fffffffe240: 0x0000000000666473      0x000000000040079d
0x7fffffffe250: 0x00007ffff7de59a0      0x0000000000000000
0x7fffffffe260: 0x0000000000400750      0x488a3f7dd9bd1f00
0x7fffffffe270: 0x00007fffffffe360      0x0000000000000000
0x7fffffffe280: 0x0000000000400750      0x00007ffff7a05b97
0x7fffffffe290: 0x0000000000000001      0x00007fffffffe368
0x7fffffffe2a0: 0x0000000100008000      0x0000000000400657
0x7fffffffe2b0: 0x0000000000000000      0x50d55bbaf0a02160
0x7fffffffe2c0: 0x0000000000400570      0x00007fffffffe360
0x7fffffffe2d0: 0x0000000000000000      0x0000000000000000
0x7fffffffe2e0: 0xaf2aa4c53b202160      0xaf2ab47a489e2160
0x7fffffffe2f0: 0x00007fff00000000      0x0000000000000000
```

使用 continue 命令，由于后续没有断点，程序将会一直运行直到结束。

代码 4: 运行结束

```
gdb-peda$ c
Continuing.
error flag
[Inferior 1 (process 14182) exited normally]
Warning: not running
gdb-peda$
```

4 实验环境搭建

本实验中会有 32 位和 64 位的程序，如果你使用的虚拟机是 64 位需要额外配置 32 位运行环境

代码 5: 32 位环境搭建

```
sudo apt-get update
sudo apt-get install libc6:i386
```

pwntools 安装

代码 6: pwntools 安装

```
sudo apt-get install python python-pip python-dev git libssl-dev libffi-dev build-
    essential
sudo pip install pwntools
```

5 实验内容

lab1 包含 4 个题目，需要对 4 个程序进行逆向。逆向结果为一个 flag 开头的可见字符串，正则形式为：flag{[0-9a-zA-Z]+}。

其中 lab1-1 是简单的逆向入门用于熟悉软件和环境，lab1-2 和 lab1-3 涉及到数学计算，需要通过基本的数学运算性质进行逆计算得出 flag，lab1-4 需要对原有的二进制进行一些修改，之后才能进行正常的输入输出。

每个程序在输入正确的 flag 后会显示 good，如果 flag 不正确将会输出 error flag。

代码 4：验证逆向结果

```
$ ./lab1-1
123
error flag
$ ./lab1-1
flag{XXXXXXXXXXXXX}
good!
$
```