武汉大学国家网络安全学院

课程作业报告

题 目： 软件安全实验

专业(班)： 信安5班

学 号： 20222302181161

姓 名： 王亚鹏

课程名称： 软件安全实验

任课教师： 赵磊

2024年 10 月 27日

目录

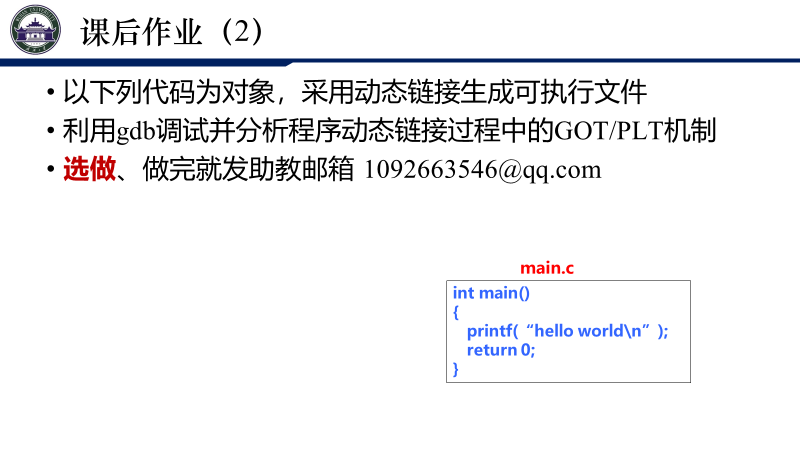
[实验内容 2](#_Toc181020950)

[实验步骤 2](#_Toc181020951)

[实验心得 5](#_Toc181020952)

[压缩包文件结构汇总 5](#_Toc181020953)

## 实验内容



## 实验步骤

**1.实验环境的准备**

经查阅资料，动态链接最好在Linux系统进行，Windows系统对某些指令不兼容或者需要晦涩的替代指令

只需要一个安装好gcc、gdb的Linux系统就可以开始进行实验

**2.main.c的动态链接**

使用gcc -o main main.c -fPIC -pie这个命令

确保在生成可执行文件时使用动态链接，这样可以通过GOT/PLT机制访问动态库函数，参数意义如下：

 -o main：将可执行文件命名为main

 -fPIC：生成与位置无关的代码

 -pie：生成位置无关的可执行文件

**3.gdb调试过程**

启动并加载可执行文件——gdb main

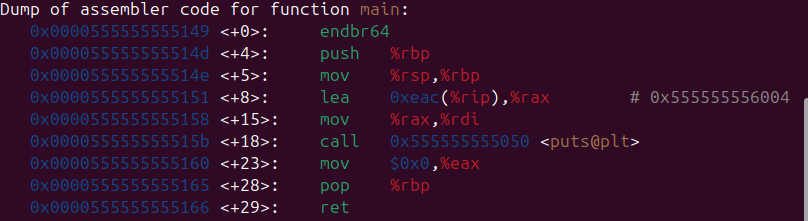
设置断点在main函数的入口——break main

启动程序——run

当程序在main函数处停止时，单步执行到调用printf函数的指令，并检查GOT（全局偏移表）和PLT（过程链接表）

Step指令用于单步执行，disassemble main用于生成反汇编代码

结果如下:



看到类似call printf@plt的指令，这表示通过PLT跳转到printf函数

查看GOT和PLT地址

info functions printf指令用于检查GOT中与printf相关的入口，我把输出（需要打开gdb的logging功能）写入了**output.txt**中，更易查看

这个输出文件中有很多与printf相关的函数实现，类似于：

* int \_\_fprintf(FILE \*, const char \*, ...) 在 stdio-common/fprintf.c
* int \_\_vprintf(const char \*, struct \_\_va\_list\_tag \*) 在 stdio-common/vprintf.c

这些函数表示了printf的不同变体以及它们的内部实现。比如，\_\_vprintf表示变长参数版本的printf，用于内部处理变长参数

**重点关注GOT/PLT机制**： 在动态链接过程中，程序通过PLT（过程链接表）跳转到GOT（全局偏移表）中的地址，以访问共享库中的函数。例如，当程序调用printf时，首先会通过PLT间接调用GOT中存储的地址，而GOT最初会指向动态链接器。通过gdb可以看到：

* printf是通过PLT入口跳转的。
* info functions命令输出的各种实现（例如vprintf, \_\_fprintf）实际都是为了支持标准库函数的不同调用形式

课上讲到的两种连接方式：

**（1）静态链接**

**1. 静态链接（Static Linking）**

**概念：**

静态链接是在**编译期**将所有外部库的代码直接复制到生成的可执行文件（如 .exe 文件）中。换句话说，编译器会将程序所需的外部库的函数、数据等完整地嵌入到最终生成的可执行文件里。

**工作流程：**

* 编译器和链接器在编译过程中，将所有程序的源代码及其依赖的库代码整合在一起，生成一个完整的可执行文件。
* 一旦生成，这个可执行文件就不再需要外部的库文件，能够独立运行。

**优点：**

* **独立性强**：因为所有的依赖都已经包含在可执行文件中，生成的 .exe 文件可以在没有其他库文件的情况下运行。这对于分发和部署非常方便。
* **加载速度快**：由于所有代码都已经包含在可执行文件中，操作系统不需要额外加载外部的库，程序启动时可以更快。

**缺点：**

* **占用空间大**：静态链接会将所有依赖库的代码都复制到可执行文件中，即使这些库中的部分功能并没有实际使用。这会导致生成的 .exe 文件体积较大。
* **难以更新**：如果外部库有更新或修复了安全漏洞，需要重新编译并分发整个可执行文件。

**应用场景：**

静态链接常用于嵌入式系统、独立发行的应用程序，或者**对性能要求很高**的场景，**因为它避免了运行时加载外部库的开销**。

**（2）动态链接**

**概念：**

动态链接是在**运行时**才将外部库链接到可执行文件。也就是说，程序编译后生成的 .exe 文件不会包含外部库的实际代码，而是依赖于系统在运行时加载这些库（通常是 .dll 文件或共享库）。

**工作流程：**

* 在编译过程中，链接器不会将依赖库的代码复制到可执行文件中，而是记录下这些库的名称和在运行时需要调用的函数。
* 当程序启动时，操作系统会根据可执行文件中的导入表（Import Table）加载相应的动态链接库（DLL）文件，将需要的函数和数据映射到程序的内存空间中。

**优点：**

* **占用空间小**：动态链接的可执行文件不会包含外部库的代码，只是记录了库的引用，因此 **.exe 文件体积较小**。
* **库的共享和更新更方便**：多个程序可以共用同一个动态库，避免了重复加载相同的代码。如果动态库被更新，所有依赖它的程序都可以立即受益于新的功能或安全修复，无需重新编译。

**缺点：**

* **依赖环境**：程序运行时必须确保所依赖的动态库存在于目标系统中。如果动态库缺失或版本不兼容，程序可能无法启动（例如，常见的“缺少 .dll 文件”错误）。
* **加载开销**：程序运行时需要操作系统动态加载外部库，这会增加启动时间和运行时的复杂性。
* **库版本管理问题**：如果不同的程序依赖相同的动态库的不同版本，可能会导致“DLL地狱”问题，即版本冲突使得程序无法正常运行。

**应用场景：**

动态链接广泛应用于桌面应用程序、服务端程序和现代操作系统中的大部分软件。由于动态库可以被多个程序共享，它非常适合那些需要降低文件体积或频繁更新的程序

## 实验心得

通过调试 main.c 代码，深入学习了动态链接的工作机制及其在 Linux 系统上的实现。实验过程中，尤其是使用 gdb 进行动态链接调试，更加直观地理解了 GOT/PLT（全局偏移表和过程链接表）的运行原理

通过使用 gdb 进行调试，我设置了断点并逐步执行代码，特别是检查了调用 printf 函数的指令。让我印象深刻的是通过 call printf@plt 的指令跳转到 PLT 表中的地址，然后再通过 GOT 表定位 printf 的实际实现。这种过程不仅清晰地展示了动态链接的延迟绑定特性，还让我对共享库的工作机制有了更深刻的认识。

在实验过程中，我还将 gdb 输出的信息记录到 output.txt 文件中，方便后续的分析。通过对输出文件的整理，我发现系统中有很多与 printf 相关的变体函数，例如 \_\_vprintf 和 \_\_fprintf，它们分别处理变长参数和文件输出等功能。让我认识到，动态链接的优势不仅在于节省空间，还在于通过延迟加载和共享库功能提高了代码的可维护性和灵活性。

通过对静态链接和动态链接两种方式的对比，我更加理解了它们的应用场景和优缺点。静态链接的独立性和加载速度优势，使其适用于对环境独立性要求高的场合，但缺点是可执行文件体积大、难以更新。而动态链接则以更小的文件体积和共享库的灵活性著称，特别是在程序更新和减少存储占用方面优势明显，但也增加了对运行环境的依赖性和复杂度

## 压缩包文件结构汇总

main.c: 进行动态链接调试的源程序

main.exe: 链接完成后的可执行文件

output.txt：GOT中与printf相关的入口汇总

markdown文件：实验中用到的导入表、导出表、两种连接方式的资料