为了顺利完成本次作业，你需要了解如下知识概念：

1. C语言格式化字符串的基本用法
2. x64 架构下程序的栈帧结构，不同类型变量在栈上的存储
3. 格式化字符串漏洞原理
4. gdb的基本命令

你可以通过回顾课上内容或者阅读拓展资料进一步复习格式化字符串漏洞的原理。

知识点

确定可控offset

利用格式化字符串的第一步是要确定我们控制的字符串在栈上的位置，只有确定后才能够进一步利用漏洞完成泄露和覆盖。

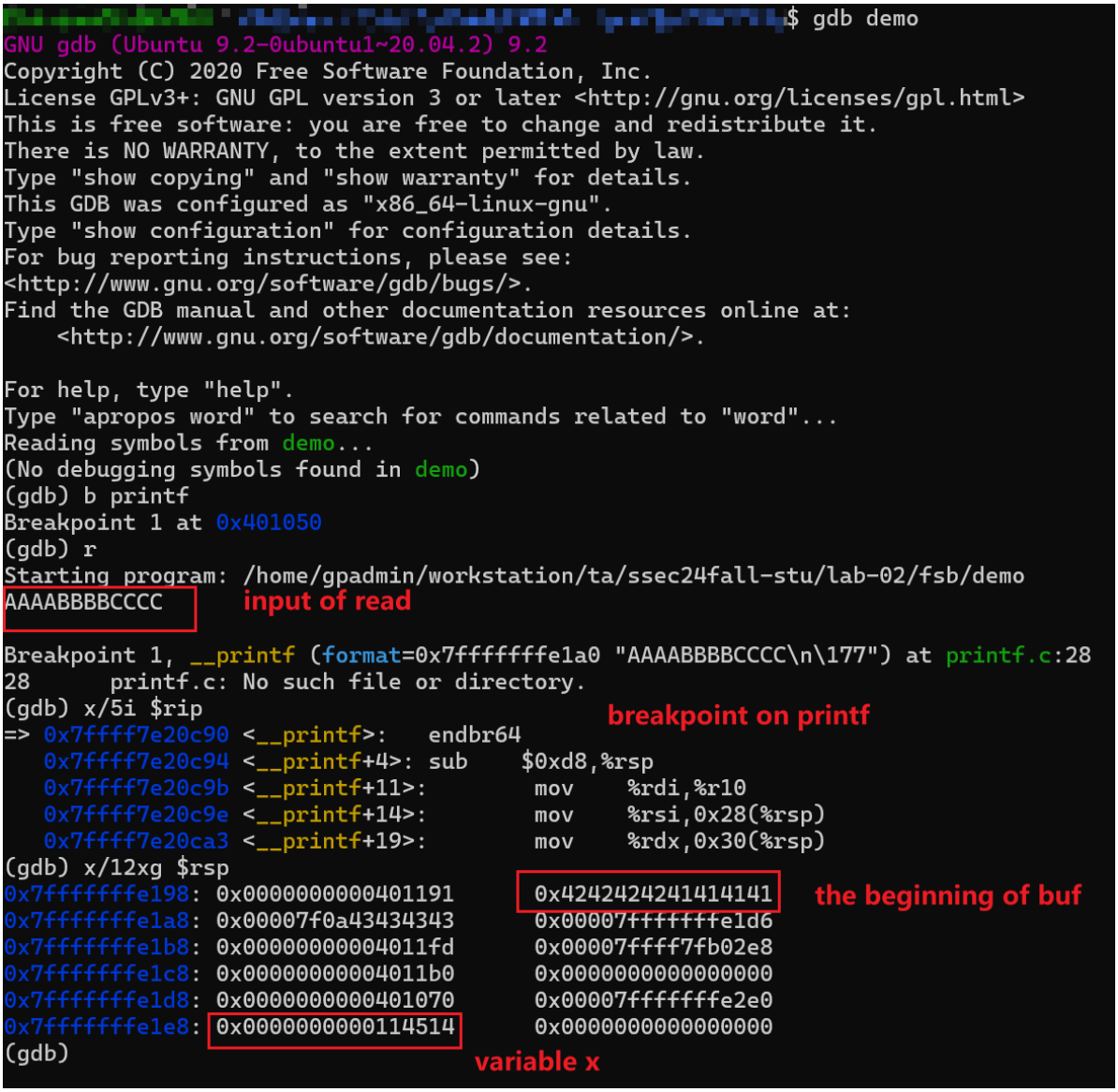
这里的可控offset是指：在进入printf函数时，栈上某个值的位置相对于格式化字符串的偏移。

查看代码`demo.c`：

```
#include <stdio.h>
#include <unistd.h>
int var = 0x1234;

int main()
{
    long long x = 0x114514;
    char buf[64];
    read(0, buf, 64);
    printf(buf);
    printf("var = 0x%x\n", var);
    return 0;
}
```

我们可以通过将断点打在`printf`处，查看触发断点时buf在栈上的位置进而确定字符串的偏移：



如图可知，根据linux下64位程序的调用规范，栈上的第一个元素是程序执行`CALL`保存的返回地址（即0x7fffffffe198指向的位置），而从0x7fffffffe1a0开始为局部变量buf。故而可推断出buf的偏移为6（除开rdi寄存器指向格式化字符串本身，rsi是第一个参数... r9是第五个参数，0x7fffffffe1a0指向的值是第六个参数）；同理我们还可推断出x的偏移为15。

另一种更常用的方式依赖如下指令：

```
$ echo "AAAAAAAA%p.%p.%p.%p.%p.%p.%p.%p" | ./demo
AAAAAAAA0x7fff9133f6b0.0x40.0x7fb0a34391f2.
(nil).0x7fb0a3546d60.0x4141414141414141.0x70252e70252e7025.0x252e70252e70252e
??Q??var = 0x1234
```

> 如果不理解代码，可以进入 lab-00 的 wiki 阅读交互部分；简言之，上面的代码将 `echo` 的输出，通过管道，重定向到 `demo` 程序的输入

简单数一下，会发现 `0x4141414141414141`，也就是我们放到 `buf` 起始位置的字符串，位于第 6 个打印位置，从而可知格式化串的偏移为 6。

泄露栈上内存

在确定偏移位置后，我们可以利用格式化字符串来泄露栈上变量的值，比如demo中变量x的值：

```
$ echo '%15$p' | ./demo
0x114514
var = 0x1234
```

更重要的是，还可以利用它来泄露`main`的返回地址，即libc 代码中的 `__libc_start_main`。

> 要了解`glibc`下程序的启动流程，可以阅读 《how glibc calls main》

在了解main函数汇编里栈帧的分配情况后，我们可以通过如下方式来泄露返回地址：

```
$ echo '%17$p' | ./demo
0x7f543efe0d90
var = 0x1234
```

这里我们拿到了main函数的返回地址`__libc_start_main`，它是libc中的一个函数。

在拿到libc中任意函数的地址后，我们即可利用它来确定libc的版本以及libc被加载的虚拟地址偏移，进而获得运行时刻所有libc符号的虚拟地址，这对于后续调用 `system` 等弹shell的libc函数是非常重要的。

泄露和覆盖任意内存

进一步地，我们可以利用格式化字符串来泄露和覆盖任意地址的内存。

在这里，我们提供了一个示例脚本`example.py`，它会生成两个文件`leak_var.txt`和`write_var.txt`：

1. `leak_var.txt`包含了针对demo文件的任意地址读payload：

   ```
   $ python3 example.py
   $ cat leak_var.txt | ./demo | hexdump -C
   00000000  41 41 41 41 34 12 38 40  40 76 61 72 20 3d 20 30  |AAAA4.8@@var = 
   0|
   00000010  78 31 32 33 34 0a                                 |x1234.|
   00000016
   ```

   > 41 41 41 41 之后的 34 12 即为读到的内容

2. `write_var.txt`则包含了针对demo文件的任意地址写payload：

   ```
   $ cat write_var.txt | ./demo
   ...
   ...
   var = 0x114514
   ```

请阅读并理解攻击脚本的内容，并复现上述过程。

实践

实践1

请阅读`fsb1.c`的内容，在本地和远程服务上完成攻击（要求getshell）。远程服务暴露在：

- ip：`8.154.20.109`，port：`10300`

实践2

请阅读`fsb2.c`的内容，在本地和远程服务上完成攻击（要求getshell）。远程服务暴露在：

- ip：`8.154.20.109`，port：`10301`

提示：

- 攻击步骤：
  1. 通过泄露libc函数来确定libc加载的虚拟地址，并通过计算拿到system的地址；
  2. 覆盖printf的GOT表为system；
  3. 调用printf以触发system从而getshell。
- 你可以通过学习pwntools中fmtstr 库的相关API来简化攻击流程。

bonus

请阅读`bonus.c`的内容，在本地和远程服务上完成攻击（要求getshell）。远程服务暴露在：

- ip：`8.154.20.109`，port：`10302`

提示：

- 本题目中字符串不再位于栈上，无法利用之前的方法覆盖任意地址的内存。但栈上一些敏感内存仍然可以被覆盖，比如函数执行`push rbp`保存的rbp寄存器。

实验要求

作业分数按如下内容组成，实验报告不限制格式，请尽可能提供多的步骤以方便评步骤分（当然，有能力一步到位也可以，请将攻击代码以附件形式提供）

- fsb1（40分），请完成对`fsb1`程序的攻击，通过覆盖栈上内存拿到shell，完成本地测试和远程，最终执行远程的`flag.exe`，提供下述截图证明，并以附件形式提交攻击代码

- fsb2（60分），请完成对`fsb2`程序的攻击，根据提示执行system函数拿到shell，完成本地测试和远程，最终执行远程的`flag.exe`，提供截图证明，并以附件形式提交攻击代码
- bonus（extra 20分）