编号：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验 | 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 总评 | 教师签名 |
| 成绩 |  |  |  |  |  |  |  |  |  |  |

武汉大学国家网络安全学院

课程实验(设计)报告

题 目： 格式化字符串漏洞

专业(班)： 信安5班

学 号： 2022302181161

姓 名： 王亚鹏

课程名称： 软件安全

任课教师： 赵磊

2024年 11 月 12日

**目录**

[**格式化字符串漏洞实验 1**](#_Toc182321700)

[**3.1实验名称 1**](#_Toc182321701)

[**3.2实验原理 1**](#_Toc182321702)

[**3.3实验要求 1**](#_Toc182321703)

[**3.4实验步骤 1**](#_Toc182321704)

[**3.4 实验结果 4**](#_Toc182321705)

[**3.5实验心得 4**](#_Toc182321706)

## 格式化字符串漏洞实验

### 3.1实验名称

格式化字符串

### 3.2实验原理

在 C 语言中，printf 等格式化输出函数会解析传入的格式化字符串，如果格式化字符串中的占位符由用户提供且未被验证，就会造成潜在的安全漏洞

printf 函数允许使用不同的格式化占位符来控制输出，例如 %x 输出十六进制，%s 输出字符串等。此外，printf 的一种特殊占位符 %n 可以将当前输出的字符数写入到指定的内存地址。因此，如果用户可以直接控制格式化字符串的内容，可能会导致以下两种攻击形式：

* **信息泄露**：使用 %x、%s 等格式符号，可以读取栈上和堆中的敏感数据，获取程序的内存布局和变量内容。
* **任意写入**：利用 %n 占位符，可以在内存的指定位置写入字符计数，从而修改变量的值或控制程序的执行流。

### 3.3实验要求

1）构造的文件内容（在in文件中）

2）程序接收了输入后，格式化串漏洞触发前的栈桢布局和关键变量的布局（参见实验步骤）

3）漏洞触发后的布局（参见实验步骤）

4）变量a的值是如何被重写的（参见实验原理和实验步骤）

### 3.4实验步骤

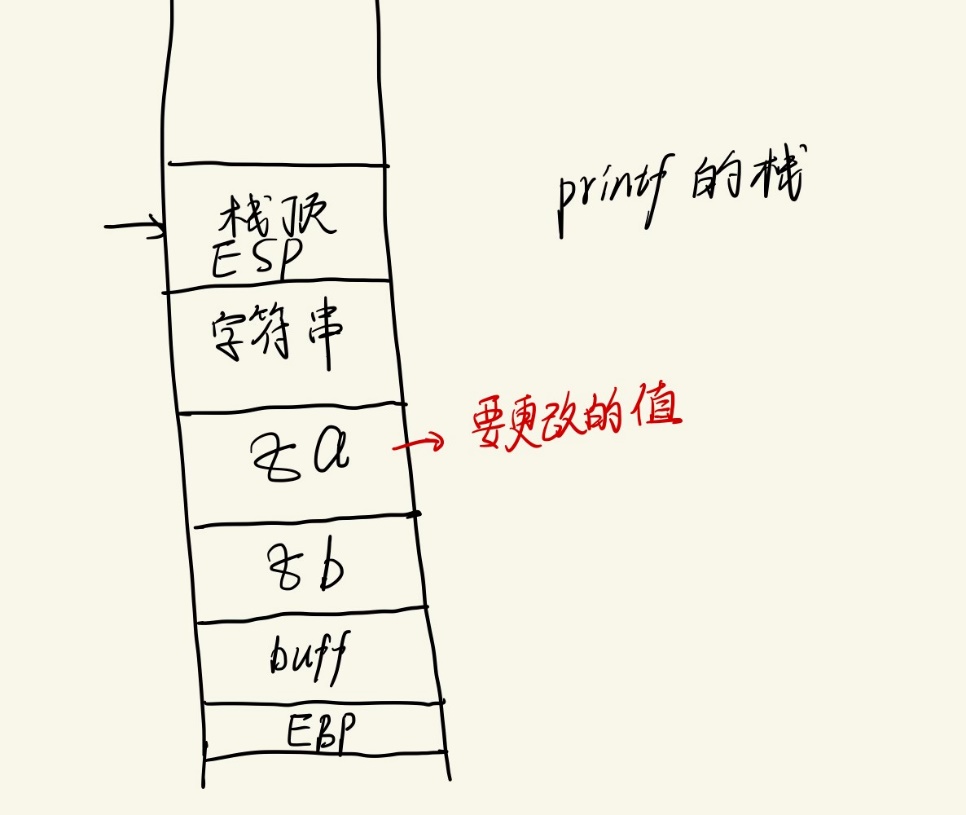
首先分析vul.c



**关键点**

1. **缓冲区定义**：buff[36] 定义了一个 36 字节的缓冲区。
2. **文件读取**：代码中使用 fread(buff, 32, 1, fp); 从文件读取最多 32 字节的数据到 buff。由于 buff 大小为 36 字节，这段代码本身不会导致缓冲区溢出问题（假设文件输入符合预期）。
3. **格式化字符串漏洞**：printf(buff); 这里使用 buff 作为 printf 的格式字符串，**没有进行格式化字符串的限制**。这意味着，如果文件中的内容包含格式化指令（例如 %x、%s 等），将导致 printf 按照这些指令执行，从而可能泄露内存信息或执行非预期的操作。
4. **变量地址泄露**：printf("the addresses of a b buf are %x,%x,%x\n", &a, &b, buff); 输出了 a、b 以及 buff 的内存地址，**暴露了堆栈布局**，可能为进一步的攻击提供信息。

14行处printf("the addresses of a b buf are %x,%x,%x\n", &a, &b, buff);对应的栈空间如下所示

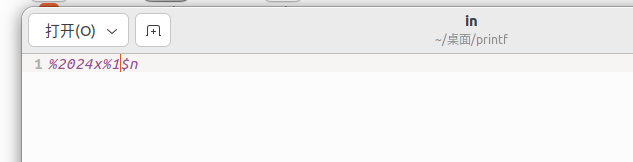


下面利用in中的内容对a的值进行修改

**方法一：**

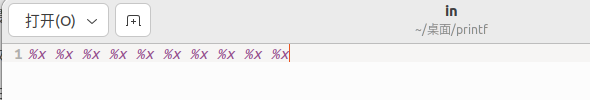
想要修改a的值，就要定位到a的地址，如上图所示，就是格式化字符串下面的位置，使用%1$n，定位到第一个格式化字符串的参数，即a所在的位置，由此可以将arg1前面输出字符串的个数写入到a中（可以参见实验原理中%n用于任意写）

想让a的值变成2024，那就让前面的输出的字符串的位数变成2024，所以使用%2024x即可



**方法二：**

还有一种方法用于确定a的位置，如下图所示：



过在 in 文件中构造一系列 %x 占位符，利用 printf 输出栈中的值，逐步试探 a 的具体位置。printf 会输出一系列十六进制值，即栈中的内容。通过观察输出结果，可以确定 a 的具体偏移位置

一旦找到 a 的栈偏移，就可以利用格式化字符串 %n 对其值进行写入。通过构造类似 %2024x%<offset>$n 的格式化字符串，可以将 2024 写入 a 的地址。

* %2024x 会输出 2024 个字符，累积字符计数至 2024。
* %<offset>$n 会将当前字符计数写入到指定的栈偏移量上

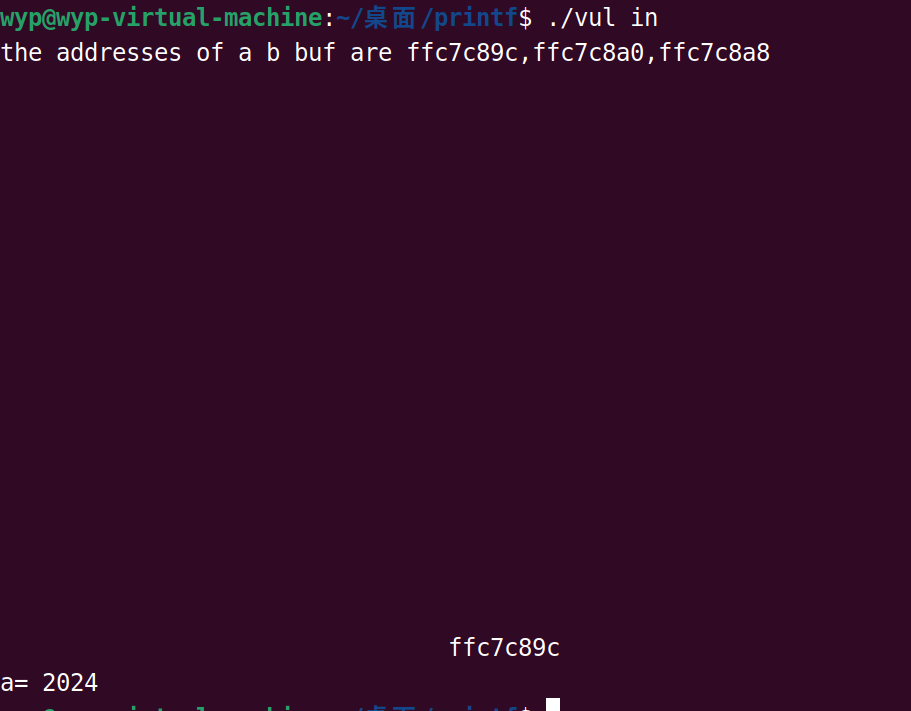
这种方法对于栈空间的分析较少，也不失为一种好方法

### 3.4 实验结果

运行时使用已经修改好的in文件

**./vul in**

运行结果如下：



可以看到a已经变成2024，按照预期运行无误

### 3.5实验心得

**1. 格式化字符串漏洞的理解与利用**

格式化字符串漏洞是一种较为隐蔽的漏洞，通常发生在程序中直接使用用户输入作为 printf 等格式化输出函数的格式化字符串时。实验中，通过构造特定的输入字符串，如 %x、%n，我们可以输出栈上数据，甚至在内存指定位置写入特定的值。在这个实验中，我们利用 %x 占位符探查栈上内容，并通过 %n 占位符实现了对变量 a 的修改。

**2. 通过格式化字符串泄露内存信息**

首先在输入文件中构造了一系列的 %x 占位符，用来观察程序栈上的内容。通过输出的十六进制数据，我们可以看到栈上的各个内存地址和值，从而逐步确认目标变量在栈中的偏移量。还有一种方法是通过分析得到a在栈中的位置。

**3. 使用 %n 实现任意内存写入**

实验的核心是利用格式化字符串中的 %n 占位符来修改变量 a 的值。具体来说，构造 %2024x%1$n 这种格式化字符串可以让 printf 输出 2024 个字符，并将该字符计数写入到指定的内存位置。通过这种方式，成功将 a 的值从 0 改为 2024，达到了实验的预期目的。

**4. 安全防范**

本次实验让我更深刻地认识到，格式化字符串漏洞虽然隐蔽，但利用起来却非常危险。在真实环境中，为了防范此类漏洞，可以采用以下几种方法：

* **永远使用格式化字符串**：始终使用固定的格式化字符串来输出用户输入的数据。例如，将 printf(buff); 改为 printf("%s", buff);，这样可以有效避免格式化字符串漏洞。
* **启用编译器保护**：现代编译器提供了诸如栈保护（Stack Protector）和地址空间布局随机化（ASLR）等安全机制，可以显著增加攻击者利用该漏洞的难度。
* **输入验证与控制**：严格控制用户输入的内容和长度，避免不受控的字符串进入 printf 等格式化输出函数