《计算机网络协议开发》实验报告

第3次实验

逆向工程套接字编程实验

姓名：陈越琦

学号：121160005

计算机系13级

邮箱：Yueqichen.0x0@gmail.com

时间: 03/16/2017

实验目的

理解协议的逆向分析方法并掌握客户端套接字编程。

实验内容

实验分成三个步骤：抓包分析，客户端实现以及服务器端实现。

抓包分析

在抓包分析环节，我通过Wireshark抓取客户端与服务器之间的交互数据，解析天气查询数据的格式，并做下原始笔记。具体过程如下：

1. 通过objdump工具分析可执行客户端文件，发现服务器的ip地址是114.212.191.33
2. 启动Wireshark，然后启动客户端，在Wireshark的filter中添加约束“ip.src==114.212.191.33”，过滤获得交互数据包
3. 发现客户端和服务器之间使用TCP连接，服务器端口为4321。在报文项右击选择“Follow TCP Stream”跟踪数据，从而流获得完整的交互过程。
4. 执行正常的交互流程：输入建议的城市，并分别使用选项1,2,3进行天气查询，获得响应的应用数据并记录。多次执行，分析记录的应用数据获得数据格式。
5. 执行边缘条件检查：输入错误的城市（随机输入字母）；在选项中输入1,2,3之外的选项；在选项3中输入不小于10的查询天数；在选项3中输入9。获得交互数据并分析，补充数据格式。
6. 整理原始笔记。
7. 画客户端状态转移图和服务器响应示意图（因为交互过程比较简单，客户端和服务器端实现直接使用了原始笔记。交互流程图和状态转移图是在撰写实验报告中完成的）。

图1是包含了客户端向服务器发送数据报文的格式的客户端状态转移图。为了描绘方便，图1省略了部分细节，比如三次握手，四次关闭以及ACK等。图2是服务器针对客户端发送的不同的数据报文做出响应的示意图。需要说明的是，图2的最后一项表示没有找到相应天气信息。客户端在收到这样的报文之后应该向用户打印提示信息。

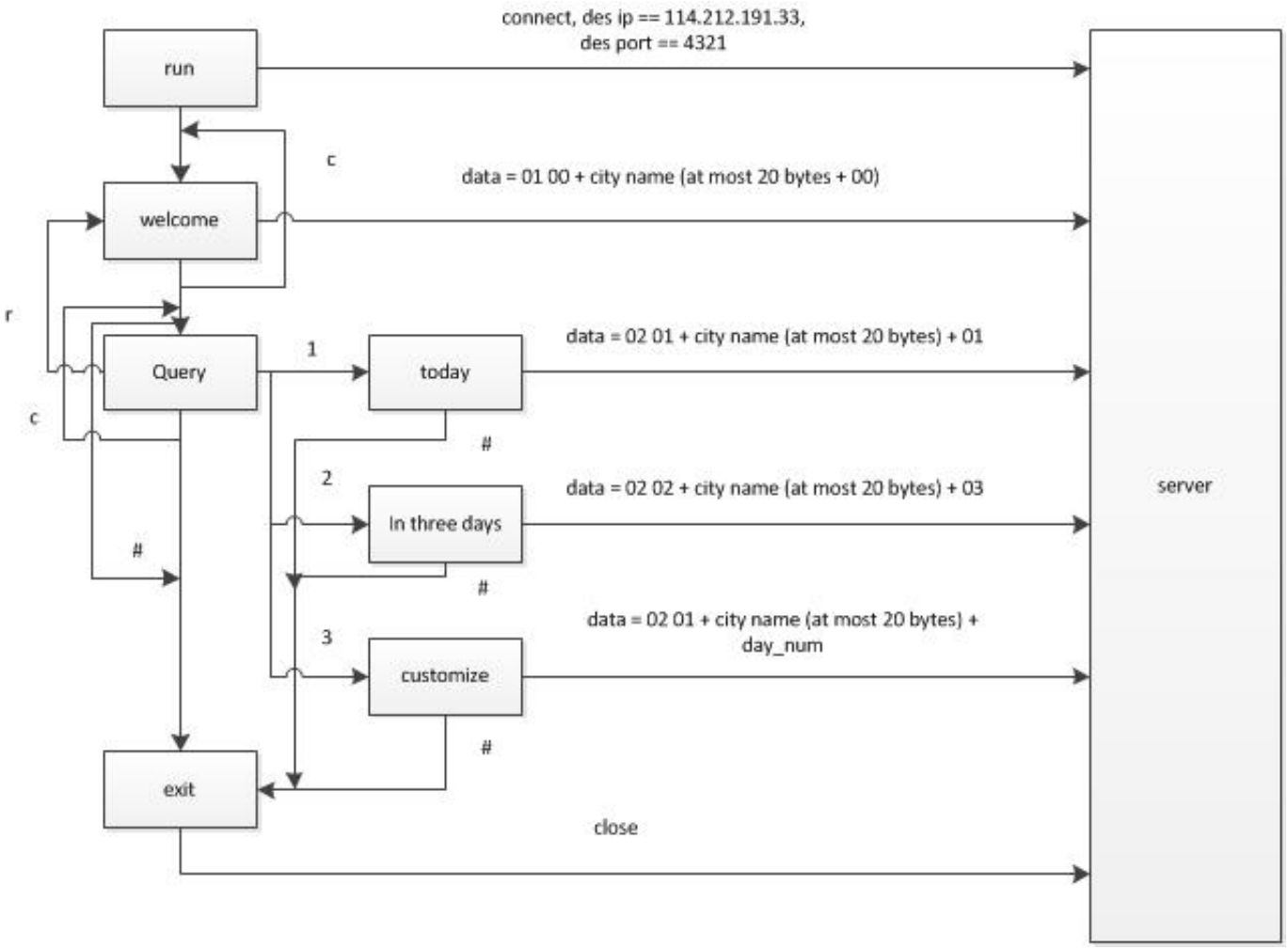
connect, des ip == 114.212.191.33,
des port == 4321
run
c
data = 01 00 + city name (at most 20 bytes + 00)
welcome
r
Query
1
today
data = 02 01 + city name (at most 20 bytes) + 01
c
#
2
In three days
data = 02 02 + city name (at most 20 bytes) + 03
server
#
#
3
data = 02 01 + city name (at most 20 bytes) +
day_num
customize
#
exit
close

图 1: 状态转移图

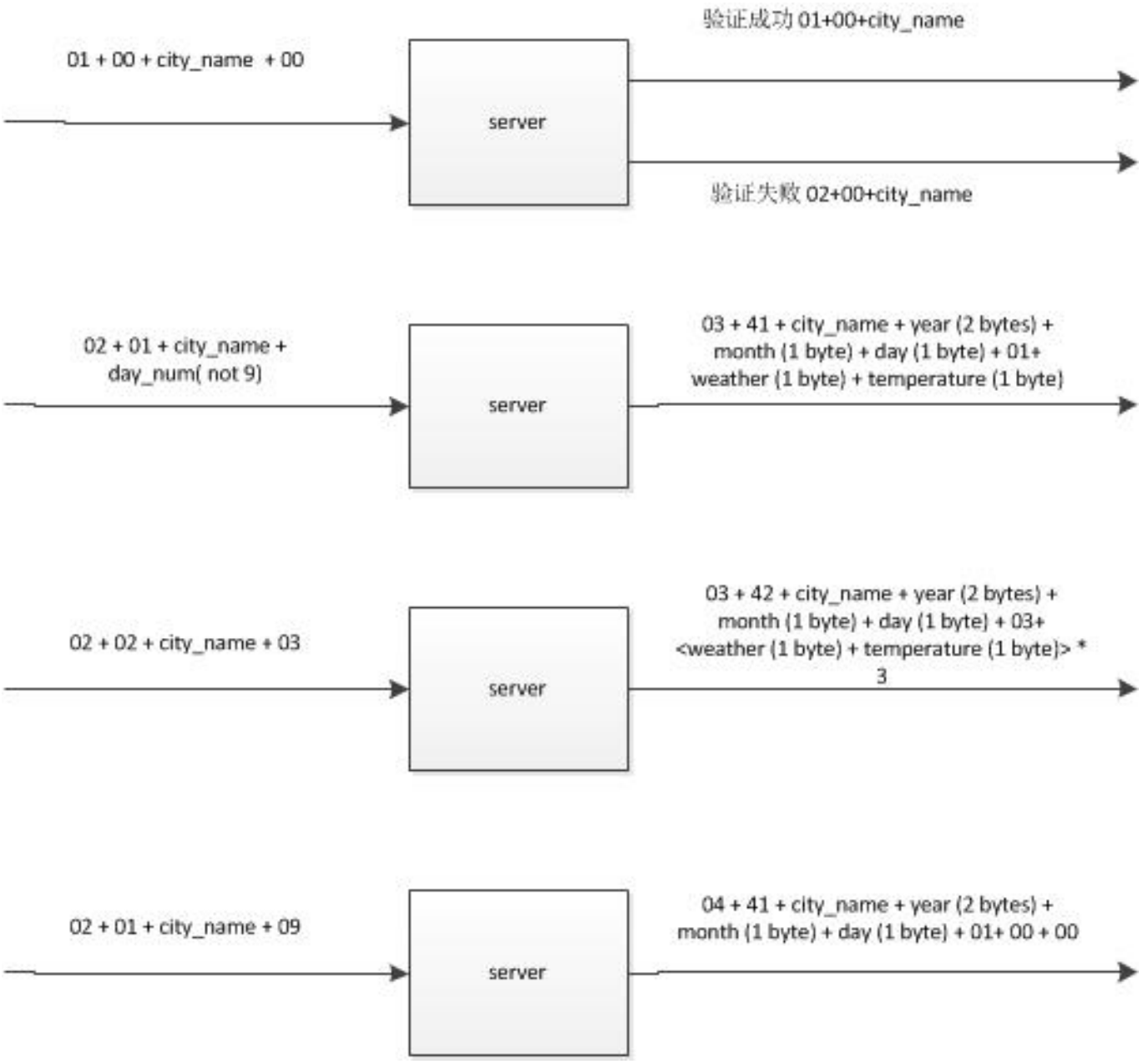

图 2: 服务器响应示意图

客户端实现

根据抓包分析结果以及[1]附录D实现客户端，并对比测试。客户端的实现中很重要的一个部分是实现一个简单的状态机。

图3是状态机定义代码，在程序启动之后进入INIT状态。程序在INIT状态中尝试与服务器建立TCP连接，连接成功之后，进入WEL状态，并打印提示信息。连接失败，则打印提示信息，并退出。

在WEL状态中，程序接受用户的输入，将每次用户输入的城市名发送到服务器进行验证，验证通过之后，就进入QUERY状态，打印提示信息。否则，程序就停留在WEL状态直到用户输入合法的城市名或者输入”#”退出。

在QUERY状态中，程序接受用户的进一步指示并在本地检查用户输入是否正确。用户输入通过检查之后，就将查询信息打包发送到服务器，服务器响应查询。客户端再对服务器的响应报文进行分析，输出打印信息。如果用户输入“#”则返回到WEL状态。

在上述的任何状态中，用户输入”#”都代表退出程序，程序会在关闭TCP连接之后结束进程。状态机的更多细节在图1中展现。

此外，客户端还有两个函数非常重要，分别是kick_out，swallow_in，对应了查询请求的数据封装及发包，和查询响应数据包的接收。（图4)。客户端发送数据和接受数据的格式在图5中定义。

```
24 enum CLIET_STATES {
25   INIT,
26   WEL,
27   QUERY,
28 };
```

图 3: 状态机定义

```
88 // send out package
89 int kick_out(unsigned char stage, unsigned char option, char *city_name, unsigned char day_num) {
90   struct send_content new_send_content;
91   new_send_content.stage = stage;
92   new_send_content.option = option;
93   strncpy(new_send_content.city_name, city_name, 20);
94   new_send_content.day_num = day_num;
95   char send_data[23]; // at most 23 bytes
96   memcpy(send_data, &new_send_content, sizeof(struct send_content));
97   return send(gsockfd, send_data, 23, 0);
98 }
99
100 // receive package
101 int swallow_in(struct recv_content *new_recv_content){
102   char recv_data[77]; // at most 77 bytes
103   recv(gsockfd, recv_data, 77, 0);
104   memcpy(new_recv_content, recv_data, sizeof(struct recv_content));
105   return 0;
106 }
```

图 4: 客户端发包收包函数

```c
#pragma pack(1)
struct send_content {
  unsigned char stage;
  unsigned char option;
  char city_name[20];
  unsigned char day_num;
};

struct recv_content {
  unsigned char res; // result of query
  unsigned char useless;
  har city_name[20];
  unsigned short year;
  unsigned char month;
  unsigned char day;
  unsigned char unit_num;
  unsigned char day1_weather;
  unsigned char day1_temp;
  unsigned char day2_weather;
  unsigned char day2_temp;
  unsigned char day3_weather;
  unsigned char day3_temp;
};
#pragma pack()
```

图 5: 数据格式

服务器端实现

根据抓包分析结果以及[1]附录D实现并发服务器端，同时启动服务器和多个客户端，对比测试。

服务器端的实现与客户端的类似，甚至更加简单（没有状态机），需要注意的是并发的实现方式。

```c
  bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
  listen(listenfd, LISTENQ);
  fprintf(stdout, "Server is running, waiting for connections ..\n");

  while (true) {
    clilen = sizeof(cliaddr);
    int connfd = accept(listenfd, (struct sockaddr*)&cliaddr, &clilen);
    fprintf(stdout, "%s\n", "Received request ...");
    if ((childpid = fork()) == 0) {
      printf("%s\n\n", "Child process created for dealing with client request");
      close(listenfd); // sub-process shall not have listen socket
      serve(connfd);
    }
    close(connfd);
  }
  return 0;
}
```

图 6: 服务器并发实现

图6中展示了服务器的的并发实现方式（与[1]附录D类似）。父进程在绑定端口之后开始

监听。一旦发现有未完成的TCP连接，就产生一个子进程去响应。在子进程中应该先把监听套接字关闭（line 163），因为子进程的主要任务是提供服务，而不是监听端口。在关闭了监听套接字之后，就转入服务例程（line 164）。在服务例程中，服务器响应客户端的查询请求：验证城市名的合法性，发送天气查询结果。

实验启示

在本次实验中，我分析了天气查询客户端与服务器的交互过程，获得相应的交互数据格式，并实现了相同功能的客户端和服务器。通过本次实验，我进一步熟悉了Wireshark工具，加深了对套接字编程的掌握，同时对实现简单状态机的过程有了更好的理解，为接下来的几次实验奠定了基础。

实验时间统计

- 抓包分析：1 hour
- 实现客户端：2.5 hours
- 实现服务器端：1 hour
- 实验报告撰写：2 hours

参考文献

[1] 计算机网络协议开发实验讲义(2017版)，南京大学计算机科学与技术系，陈健，附录D