title: Hacking with Linux networking CLI tools

author: 郭建平 (20231159042)

date: 2024-06-28

Packet analysis

```
sudo tcpdump -ilo -vvvnnxXSK -s0 port 9001
```

Upon running the above command, the following packet is captured:

```
tcpdump: listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:36:08.078031 IP (tos 0x0, ttl 64, id 20066, offset 0, flags [DF], proto TCP (6), length 64)
127.0.0.1.9001 > 127.0.0.1.56684: Flags [P.], seq 2268450165:2268450177, ack 2697067965, win 512, options [nop,nop,TS val 3101569467 ecr 3101547391], length 12
0x0000:  4500 0040 4e62 4000 4006 ee53 7f00 0001  E..@Nb@.@..S....
0x0010:  7f00 0001 2329 dd6c 8735 cd75 a0c1 fdbd  ....#).l.5.u....
0x0020:  8018 0200 fe34 0000 0101 080a b8de 31bb  .....4........1.
0x0030:  b8dd db7f 6865 6c6c 6f20 776f 726c 640a  ....hello.world.
```

1. Tell me the meaning of each option used in the previous command.
   - **-i**:指定网卡接口
   - **-nn**:不解析主机名和服务名，仅显示数字地址和端口
   - **-vvv**:提供最详细的输出信息。
   - **-x**:显示数据包的十六进制内容。
   - **-X**:同时以十六进制和ASCII格式显示数据包的内容。
   - **-S**:显示绝对的序列号。
   - **-K**:不校验数据包的校验和。
   - **-s0**:捕获整个数据包，而不仅仅是前96字节。
2. Please analyze this captured packet and explain it to me as detailed as you can.
   - **Answer:**
   - 时间戳：20:36:08.078031
   - 源IP和端口：127.0.0.1.9001
   - 目标IP和端口：127.0.0.1.56684
   - 协议：TCP
   - 标志：P（推送）和.（确认）
   - 序列号范围：2268450165至2268450177（长度12字节）

- 确认号：2697067965
- 窗口大小：512
- 选项：时间戳（TS val和ecr）
- 载荷内容（ASCII）：`hello world`

HTTP

1. Write a simple script showing how HTTP works (you need `curl`).

```
#!/bin/bash

# 使用 curl 发送一个 HTTP GET 请求
echo "Sending HTTP GET request..."
curl -v https://www.swfu.edu.cn/

# 使用 curl 发送一个 HTTP POST 请求
echo "Sending HTTP POST request..."
curl -v -X POST -d "action=get_captcha" https://0x00.fun/wp-admin/admin-ajax.php

# 使用 curl 发送一个 HTTP 请求并保存响应到文件
echo "Saving HTTP response to file..."
curl -v -o response.txt https://www.baidu.com

# 使用 curl 获取 HTTP 响应头
echo "Fetching HTTP headers..."
curl -v -I https://www.baidu.com
```

2. Record your HTTP demo session with `ttyrec`.

Socket programming

TCP

```
  /* A simple TCP server written in C */

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

#define PORT 65534
#define BACKLOG 5 // 允许的连接请求队列长度
#define BUFFER_SIZE 1024

int main() {
    int sockfd, new_sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len;
    char buffer[BUFFER_SIZE];
    int recv_len;

    // 创建 TCP 套接字
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    // 设置服务器地址结构
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET; // IPv4 地址族
    server_addr.sin_addr.s_addr = INADDR_ANY; // 使用 INADDR_ANY 表示接收所有网卡的连
接
    server_addr.sin_port = htons(PORT); // 端口号，需要使用 htons() 转换为网络字节顺序

    // 绑定套接字到服务器地址
    if (bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    // 监听连接请求
    if (listen(sockfd, BACKLOG) < 0) {
        perror("listen failed");
        exit(EXIT_FAILURE);
    }

    printf("Server is listening on port %d...\n", PORT);

    while (1) {
        // 接受客户端连接请求
        client_len = sizeof(client_addr);
        new_sockfd = accept(sockfd, (struct sockaddr*)&client_addr, &client_len);
        if (new_sockfd < 0) {
            perror("accept failed");
            exit(EXIT_FAILURE);
        }

        printf("Accepted connection from %s:%d\n",
               inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));

        while (1) {
            // 接收数据
            recv_len = recv(new_sockfd, buffer, sizeof(buffer), 0);
            if (recv_len > 0) {
                buffer[recv_len] = '\0';
                printf("Received: %s\n", buffer);

                // 转换为大写
                for (int i = 0; buffer[i] != '\0'; ++i) {
                    buffer[i] = toupper(buffer[i]);
                }
                // 发送大写后的数据
                if (send(new_sockfd, buffer, strlen(buffer), 0) < 0) {
                    perror("send failed");
                    break;
                }
            } else if (recv_len == 0) {
                printf("Client closed the connection\n");
                break;
            } else {
                perror("recv failed");
                break;
            }
        }

        // 关闭本次连接
        close(new_sockfd);
    }

    close(sockfd);
    return 0;
}
```

```
  /* A simple TCP client written in C */

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

#define PORT 65534
#define BUFFER_SIZE 1024

int main() {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE];

    // 创建 TCP 套接字
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    // 设置服务器地址结构
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET; // IPv4 地址族
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); // 服务器 IP 地址
    server_addr.sin_port = htons(PORT); // 服务器端口号，需要使用 htons() 转换为网络字
节顺序

    // 连接服务器
    if (connect(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
{
        perror("connection failed");
        exit(EXIT_FAILURE);
    }

    printf("Connected to server on port %d\n", PORT);

    while (1) {
        // 发送数据
        printf("Input lowercase sentence (type 'exit' to quit): ");
        if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
            perror("fgets error or EOF");
            break;
        }

        // 去除末尾的换行符
        buffer[strcspn(buffer, "\n")] = '\0';

        if (strcmp(buffer, "exit") == 0) {
            break;
        }

        if (send(sockfd, buffer, strlen(buffer), 0) < 0) {
            perror("send failed");
            break;
        }

        // 接收数据
        int recv_len = recv(sockfd, buffer, sizeof(buffer) - 1, 0);
        if (recv_len < 0) {
            perror("recv failed");
            break;
        } else if (recv_len == 0) {
            printf("Server closed the connection\n");
            break;
        }

        buffer[recv_len] = '\0';
        printf("From Server: %s\n", buffer);
    }

    // 关闭连接
    close(sockfd);
    return 0;
}
```

UDP

```
  /* A simple UDP server written in C */
```

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

#define PORT 65533
#define BUFFER_SIZE 2048

int main() {
    int sockfd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len;
    char buffer[BUFFER_SIZE];
    int recv_len;

    // 创建 UDP 套接字
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    // 设置服务器地址结构
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET; // IPv4 地址族
    server_addr.sin_addr.s_addr = INADDR_ANY; // 使用 INADDR_ANY 表示接收所有网卡的连
接
    server_addr.sin_port = htons(PORT); // 端口号，需要使用 htons() 转换为网络字节顺序

    // 绑定套接字到服务器地址
    if (bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("bind failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    printf("The server is ready to receive\n");

    while (1) {
        // 接收数据
        client_len = sizeof(client_addr);
        recv_len = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct
sockaddr*)&client_addr, &client_len);
        if (recv_len < 0) {
            perror("recvfrom failed");
            close(sockfd);
            exit(EXIT_FAILURE);
        }

        buffer[recv_len] = '\0';
        printf("Received: %s\n", buffer);
```

```c
        // 如果接收到 "exit"，退出循环
        if (strcmp(buffer, "exit") == 0) {
            break;
        }

        // 转换为大写
        for (int i = 0; buffer[i] != '\0'; ++i) {
            buffer[i] = toupper(buffer[i]);
        }

        // 发送大写后的数据
        if (sendto(sockfd, buffer, recv_len, 0, (struct sockaddr*)&client_addr,
client_len) < 0) {
            perror("sendto failed");
            close(sockfd);
            exit(EXIT_FAILURE);
        }
    }

    close(sockfd);
    return 0;
}
```

```c
  /* A simple UDP client written in C */

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

#define SERVER_PORT 65533
#define BUFFER_SIZE 2048

int main() {
    int sockfd;
    struct sockaddr_in server_addr;
    char buffer[BUFFER_SIZE];
    int message_len;
    socklen_t addr_len;

    // 创建 UDP 套接字
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    // 设置服务器地址结构
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(SERVER_PORT);
    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
        perror("inet_pton failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    }

    while (1) {
        // 获取用户输入
        printf("Input lowercase sentence (type 'exit' to quit): ");
        fgets(buffer, BUFFER_SIZE, stdin);
        message_len = strlen(buffer);
        if (buffer[message_len - 1] == '\n') {
            buffer[message_len - 1] = '\0'; // 去掉换行符
            message_len--;
        }

        // 发送数据到服务器
        if (sendto(sockfd, buffer, message_len, 0, (struct
sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
            perror("sendto failed");
            close(sockfd);
            exit(EXIT_FAILURE);
        }

        // 如果用户输入了 "exit"，退出循环
        if (strcmp(buffer, "exit") == 0) {
            break;
        }

        // 接收来自服务器的响应
        addr_len = sizeof(server_addr);
        int recv_len = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct
sockaddr*)&server_addr, &addr_len);
        if (recv_len < 0) {
            perror("recvfrom failed");
            close(sockfd);
            exit(EXIT_FAILURE);
        }

        buffer[recv_len] = '\0';
        printf("From Server: %s\n", buffer);
    }

    // 关闭套接字
    close(sockfd);

    return 0;
}
```

Questions

List at least 5 problems you've met while doing this work. When listing your problems, you have to tell me:

1. Description of this problem. For example,
    - What were you trying to do before seeing this problem?
2. How did you try solving this problem? For example,
    - Did you google? web links?
    - Did you read the man page?
    - Did you ask others for hints?

Problems

1. tcpdump 命令无法识别
    - 使用sudo apt-get install tcpdump命令安装
2. C 编译器缺失
    - 询问chatgpt以后，使用sudo apt-get install gcc,最终能够成功实现
3. Socket 编程中的地址绑定失败
    - 通过使用chatgpt了解设置套接字选项以允许地址重用，然后再检查端口是否被占用
4. 本地系统tty录屏问题
    - 在使用的Archlinux上面,无法安装ttyrec,所以使用虚拟机创建的centos,apt install ttyrec解决录屏问题
5. 不熟悉markdown格式
    - 通过菜鸟教程知道了基本的语法