---

title: Hacking with Linux networking CLI tools
author: 付火焱 (20231159082)
date: 2024-6-30
CJKmainfont: Noto Serif CJK SC
CJKsansfont: Noto Sans CJK SC
header-includes: |

<style>
body { min-width: 80% !important; }
</style>

Packet analysis

```sh
sudo tcpdump -ilo -nnvvvxXKS -s0 port 3333
```

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

```

16:21:56.263970 IP (tos 0x0, ttl 64, id 14296, offset 0, flags [DF], proto TCP (6), length 52)
127.0.0.1.12000 > 127.0.0.1.47380: Flags [.], seq 1215421731, ack 2338669079, win 260, options
[nop,nop,TS val 3760298245 ecr 3760298245], length 0

    - 0x0000: 4500 0034 37d8 4000 4006 04ea 7f00 0001 E..47.@.@.......

    - 0x0010: 7f00 0001 2ee0 b914 4871 dd23 8b65 4217 ........Hq.#.eB.

    - 0x0020: 8010 0104 fe28 0000 0101 080a e021 9905 .....(..!...

    - 0x0030: e021 9905 ..!..
```

1. Tell me the meaning of each option used in the previous command.

    - **-i**:通常用于指定命令或工具在哪个网络接口上执行操作。例如，在网络抓包工具（如tcpdump）中，使用**-i**选项可以指定监听的网卡接口，或者在一些网络管理工具中，可以指定发送数据的网络接口。

    - **-nn**:常用于显示数字地址和端口号，而不尝试将它们解析为名称。适用于脚本编写和查看原始IP地址和端口。

    - **-vvv**:
通常表示详细输出的高级别，用于增加命令输出的详细程度。'v'的数量可以不同（如 ‘-v’, ‘-vv’，‘-vvv’），更多的'v'表示更高的详细程度。

- **-
  x**:通常表示以十六进制格式显示输出。常用于显示数据的原始字节，特别是在数据包捕获和分析工具中。
- - **-
  X**:通常表示以十六进制格式显示数据，但可能包含特定于命令或工具的其他信息或格式化细节。
- **-
  S**:具体含义取决于使用的命令或工具。在某些情况下，它可能表示启动服务或会话，而在其他情况下，可能表示设置特定的选项或行为。
- **-K**:同样取决于上下文，通常用于终止与指定命令或会话相关联的进程或连接。
- **-s0**:通常指定与扫描相关的特定扫描类型或参数。在这种情况下，'-
  s0'通常指不设置任何TCP标志的TCP扫描，通常称为"TCP Null Scan"。

2. Please analyze this captured packet and explain it to me as detailed as you can.
   - **Answer:**
   - 时间戳：16:21:56.263970
   - IP头部信息：
     - TOS字段：0x0
     - TTL字段：64
     - ID字段：14296
     - Offset字段：0
     - 标志位：DF (Don't Fragment)
     - 协议：TCP (6)
     - 数据长度：52 bytes
   - 源IP地址：127.0.0.1，源端口：12000
   - 目标IP地址：127.0.0.1，目标端口：47380
   - TCP头部信息：
     - 标志：Flags [.]
     - 序列号：1215421731
     - 确认号：2338669079
     - 窗口大小：260
     - 选项：[nop,nop,TS val 3760298245 ecr 3760298245]
   - 数据长度：0

这是完整的Markdown格式，包含了抓包信息和详细的解析内容。

HTTP

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

```sh

```
#!/bin/bash

#URL to use for the requests

URL="http://httpbin.org"

# Perform a GET request

echo"Performing a GET request to $URL/get"

curl -v "$URL/get"# Perform a POST request

echo -e "\n\nPerforming a POST request to $URL/post with some data"

curl -v -X POST -d "name=ChatGPT&age=1""$URL/post"

#Display HTTP headers

echo -e "\n\nDisplaying HTTP headers for GET request to $URL/get"

curl -I "$URL/get"
```

```

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

Socket programming

TCP

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

// Your code
```

```
#include stdio.h
#include stdlib.h
#include string.h
#include unistd.h
#include arpa
#include sys#define PORT 12000
#define BUFFER_SIZE 1024int main() {
int server_fd, new_socket, valread;
struct sockaddr_in address;
int opt = 1;
int addrlen = sizeof(address);
char buffer[BUFFER_SIZE] = {0};
char *response = "Message received";// 创建套接字文件描述符
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
}

// 设置套接字选项，允许地址重用
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
sizeof(opt))) {
    perror("Setsockopt failed");
    exit(EXIT_FAILURE);
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// 将套接字绑定到指定端口
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

// 开始监听传入连接
if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

// 持续接受客户端连接并处理
while (1) {
    // 接受客户端连接
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    // 从客户端读取数据
    valread = read(new_socket, buffer, BUFFER_SIZE);
    printf("Received message from client: %s\n", buffer);

    // 向客户端发送回复消息
    send(new_socket, response, strlen(response), 0);
    printf("Reply sent to client: %s\n", response);

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

return 0;
// 创建套接字文件描述符
```

```c
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
}

// 设置套接字选项，允许地址重用
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt,
sizeof(opt))) {
    perror("Setsockopt failed");
    exit(EXIT_FAILURE);
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// 将套接字绑定到指定端口
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

// 开始监听传入连接
if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

// 持续接受客户端连接并处理
while (1) {
    // 接受客户端连接
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    // 从客户端读取数据
    valread = read(new_socket, buffer, BUFFER_SIZE);
    printf("Received message from client: %s\n", buffer);

    // 向客户端发送回复消息
    send(new_socket, response, strlen(response), 0);
    printf("Reply sent to client: %s\n", response);

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

return 0;
}
```

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

// Your code
```

```
#include stdio.h
#include stdlib.h
#include string.h
#include unistd.h
#include arpa
#include sys#define PORT 12000
#define BUFFER_SIZE 1024int main() {
int sock = 0, valread;
struct sockaddr_in serv_addr;
char *message = "Hello from client";
char buffer[BUFFER_SIZE] = {0};// 创建套接字
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
perror("Socket creation failed");
return -1;
}serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);// 将IPv4地址从点分十进制转换为二进制格式
if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0) {
perror("Invalid address/ Address not supported");
return -1;
}// 连接服务器
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
perror("Connection failed");
return -1;
}// 向服务器发送消息
send(sock, message, strlen(message), 0);
printf("Message sent to server: %s\n", message);// 从服务器接收回复
valread = read(sock, buffer, BUFFER_SIZE);
printf("Message from server: %s\n", buffer);return 0;
}
~~~

```c

## UDP

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

// Your code
~~~

```
#include stdio.h
#include stdlib.h
#include string.h
#include unistd.h
#include arpa#define PORT 12000
#define MAXLINE 1024int main() {
int sockfd;
char buffer[MAXLINE];
struct sockaddr_in servaddr, cliaddr;// 创建UDP套接字
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
```

```
}

memset(&servaddr, 0, sizeof(servaddr));
memset(&cliaddr, 0, sizeof(cliaddr));

// 设置服务器地址结构
servaddr.sin_family = AF_INET; // IPv4
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(PORT);

// 绑定服务器地址
if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

int len, n;
char *response = "Message received";

// 持续接收和回复客户端消息
while (1) {
    len = sizeof(cliaddr);

    // 接收来自客户端的消息
    n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr
*)&cliaddr, &len);
    buffer[n] = '\0';
    printf("Received message from client: %s\n", buffer);

    // 发送回复给客户端
    sendto(sockfd, (const char *)response, strlen(response), MSG_CONFIRM, (const
struct sockaddr *)&cliaddr, len);
    printf("Reply sent to client: %s\n", response);
}

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

memset(&servaddr, 0, sizeof(servaddr));
memset(&cliaddr, 0, sizeof(cliaddr));

// 设置服务器地址结构
servaddr.sin_family = AF_INET; // IPv4
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(PORT);

// 绑定服务器地址
if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
    perror("Bind failed");
    exit(EXIT_FAILURE);
```

```
}

int len, n;
char *response = "Message received";

// 持续接收和回复客户端消息
while (1) {
    len = sizeof(cliaddr);

    // 接收来自客户端的消息
    n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr
*)&cliaddr, &len);
    buffer[n] = '\0';
    printf("Received message from client: %s\n", buffer);

    // 发送回复给客户端
    sendto(sockfd, (const char *)response, strlen(response), MSG_CONFIRM, (const
struct sockaddr *)&cliaddr, len);
    printf("Reply sent to client: %s\n", response);
}

return 0;
}
```

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

// Your code
```

```
#include stdio.h
#include stdlib.h
#include string.h
#include unistd.h
#include arpa#define PORT 12000
#define MAXLINE 1024int main() {
int sockfd;
char buffer[MAXLINE];
struct sockaddr_in servaddr;// 创建UDP套接字
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
}

memset(&servaddr, 0, sizeof(servaddr));

// 设置服务器地址结构
servaddr.sin_family = AF_INET; // IPv4
servaddr.sin_port = htons(PORT);
servaddr.sin_addr.s_addr = INADDR_ANY;

char *message = "Hello from client";

// 发送消息给服务器
sendto(sockfd, (const char *)message, strlen(message), MSG_CONFIRM, (const struct
sockaddr *)&servaddr, sizeof(servaddr));
printf("Message sent to server: %s\n", message);

int n, len;

// 接收服务器的回复
n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr
*)&servaddr, &len);
buffer[n] = '\0';
printf("Message from server: %s\n", buffer);

close(sockfd);

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

memset(&servaddr, 0, sizeof(servaddr));

// 设置服务器地址结构
servaddr.sin_family = AF_INET; // IPv4
servaddr.sin_port = htons(PORT);
servaddr.sin_addr.s_addr = INADDR_ANY;

char *message = "Hello from client";

// 发送消息给服务器
sendto(sockfd, (const char *)message, strlen(message), MSG_CONFIRM, (const struct
sockaddr *)&servaddr, sizeof(servaddr));
printf("Message sent to server: %s\n", message);

int n, len;

// 接收服务器的回复
n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr
*)&servaddr, &len);
buffer[n] = '\0';
printf("Message from server: %s\n", buffer);

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?

    - I was trying to convert a README.md file to a PDF format using pandoc.

2. How did you try solving this problem? For example,
    - Did you google? web links?
    - Yes, I googled for solutions and found information about using pandoc with xelatex for converting Markdown to PDF while handling Unicode characters.
    - Did you read the man page?
    - Yes, I read the manual pages for pandoc and xelatex to understand how to address Unicode character issues.
    - Did you ask others for hints?
    - Yes, I asked for help on forums to see if others had encountered a similar problem and how they resolved it.

Problems

3. 命令不熟悉
4. 对网络知识有欠缺
5. 写脚本时遇到困难，导致失败
6. 对c语言的掌握还不够
7. 抓包需要提取权限