# 网络安全实践课程实验报告

网络端口扫描系统

张一白

161320224

目录

[网络安全实践课程实验报告 1](#_Toc454485221)

[1. 实验要求 3](#_Toc454485222)

[2. 实验环境 3](#_Toc454485223)

[3. 系统设计与实现 3](#_Toc454485224)

[4. 使用说明 5](#_Toc454485225)

[5. 程序运行截图 5](#_Toc454485226)

[6. 源代码 5](#_Toc454485227)

## 实验要求

采用socket，设计并实现一个网络扫描器，能够检查给定IP地址范围的主机端口状态。

## 实验环境

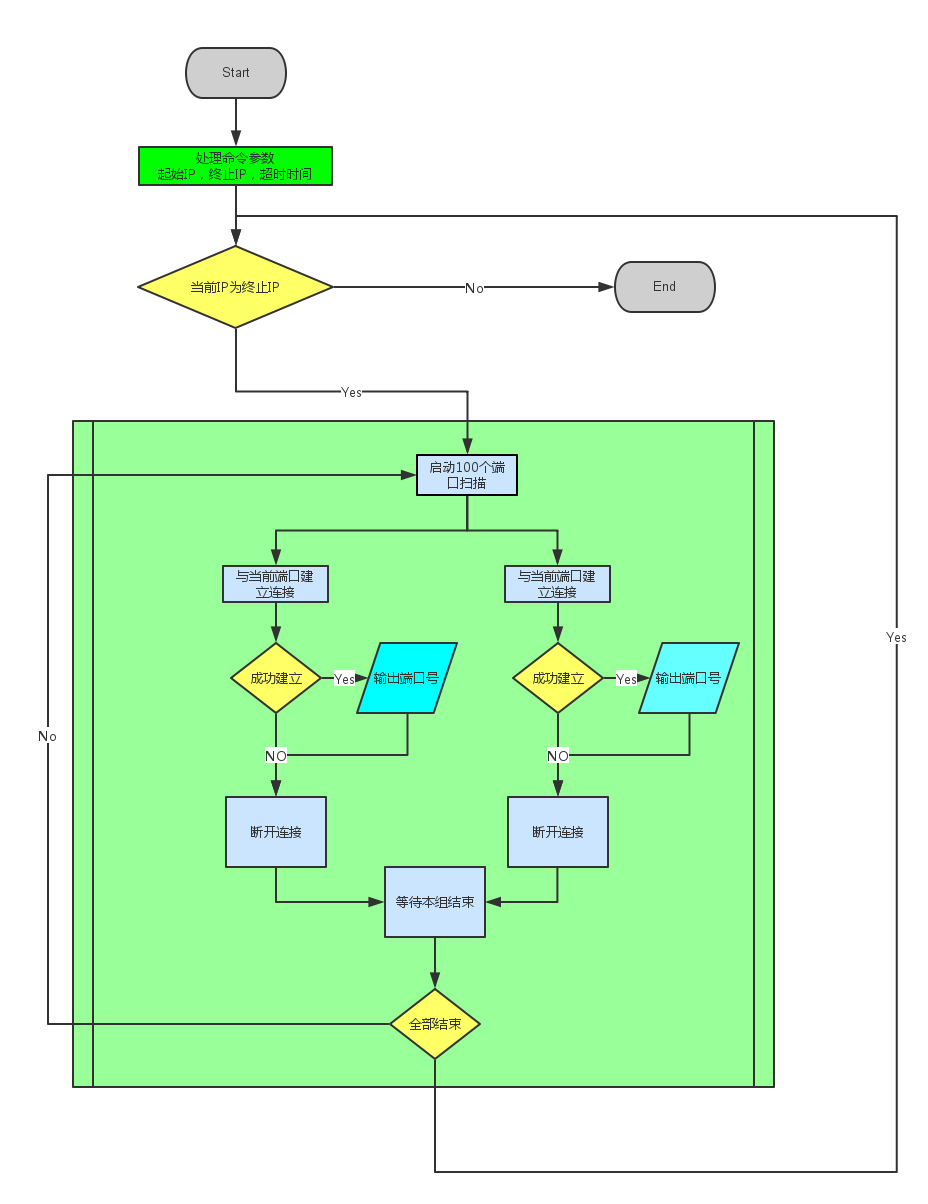
Darwin Summers-MBP.lan 15.5.0 Darwin Kernel Version 15.5.0: Tue Apr 19 18:36:36 PDT 2016; root:xnu-3248.50.21~8/RELEASE\_X86\_64 x86\_64

Linux Ubuntu-Dev 4.4.0-24-generic #43-Ubuntu SMP Wed Jun 8 19:27:37 UTC 2016 x86\_64 x86\_64 x86\_64 GNU/Linux

## 系统设计与实现

（重点流程图+说明）

采用TCP Connect方式多线程扫描，如果能与目标端口成功建立连接则目标端口开放。



## 使用说明

编译：

clang main.c –o scanner –O2 -lpthread (Mac OS X/Linux/BSD with clang)

gcc main.c –o scanner –O2 –lpthread (Mac OS X/Linux/BSD with gcc)

运行:

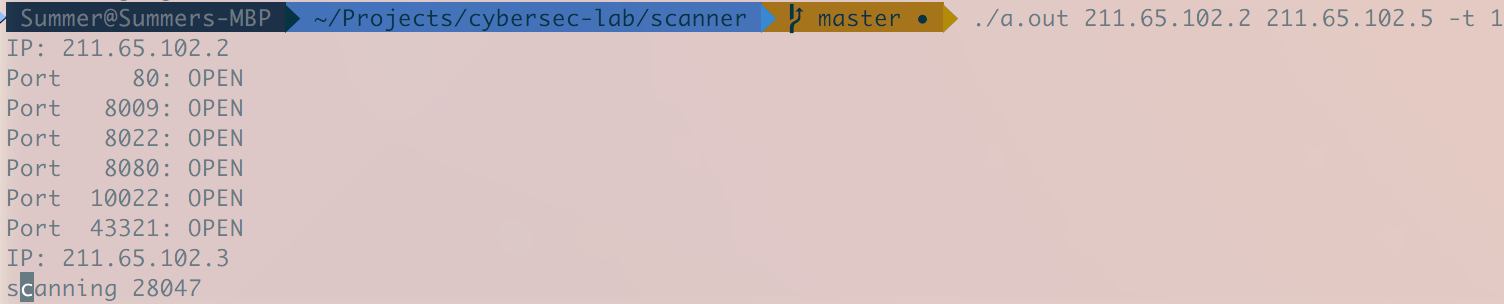
Usage: ./scanner start\_ip end\_ip [options]

-t n

--timeout=n Set timeout to n, default: 2 (sec).

如 ./scanner 211.65.102.2 211.65.102.5 –t 1

## 程序运行截图





## 源代码

|  |
| --- |
| //  // main.c  // scanner  //  // Created by Summer on 6/9/16.  // Copyright © 2016 summer. All rights reserved.  //  #include <unistd.h>  #include <sys/types.h> // socket  #include <sys/socket.h> // socket  //#include <sys/ioctl.h> // ioctl  #include <sys/fcntl.h>  #include <net/if.h> // ifreq  #include <netinet/tcp.h> // tcp  #include <netinet/in.h> // tcp  #include <arpa/inet.h>  #include <string.h> // strcpy  #include <stdio.h> // printf  #include <errno.h>  #include <pthread.h>  int do\_connect\_scan(in\_addr\_t addr, int port);  void set\_timeout(int sec);  int scan\_ip(unsigned int ip);  int print\_help(char \*argv0);  void \*connect\_scan(void \*args);  struct timeval tout;  typedef struct {  in\_addr\_t addr;  int tgt\_port;  int result;  } scan\_args;  int main(int argc, const char \* argv[]) {  unsigned int start, end, i, j, k, timeout = 2;  if(argc < 3)  return print\_help(argv[0]);  start = ntohl(inet\_addr(argv[1]));  end = ntohl(inet\_addr(argv[2]));  if (i == INADDR\_NONE || j == INADDR\_NONE)  return print\_help(argv[0]);  else if(i>j)  return print\_help(argv[0]);  for (i=3; i<argc; ++i)  {  if(strstr(argv[i], "-t") == argv[i])  if(i +1 < argc)  {  if(EOF == sscanf(argv[i + 1], "%d", &timeout))  return print\_help(argv[0]);  }  else  return print\_help(argv[0]);  else if(strstr(argv[i], "--timeout="))  if(EOF == sscanf(argv[i], "--timeout=%d", &timeout))  return print\_help(argv[0]);  }  set\_timeout(timeout);  for(k = start; k <= end; ++k)  {  scan\_ip(htonl(k));  }  return 0;  }  int scan\_ip(in\_addr\_t addr)  {  int i, j;  scan\_args args\_list[100];  pthread\_t pth\_list[100];  struct in\_addr i\_addr;  i\_addr.s\_addr = addr;  printf("IP: %s \n", inet\_ntoa(i\_addr));  for(i=0; i<600; ++i)  {  for(j = 0; j<100; ++j)  {  args\_list[j].addr = addr;  args\_list[j].tgt\_port = i \* 100 + (j + 1);  printf("scanning %d\r",args\_list[j].tgt\_port);  fflush(stdin);  pthread\_create(pth\_list + j, NULL, connect\_scan, args\_list + j);  }  for(j = 0; j<100; ++j)  {  pthread\_join(pth\_list[j], NULL);  if(args\_list[j].result)  printf("\rPort%7d: OPEN\n", args\_list[j].tgt\_port);  }  }  return 0;  }  int print\_help(char \*argv0)  {  printf("Usage: %s start\_ip end\_ip [options]\n", argv0);  printf("\t-t n\n");  printf("\t--timeout=n\t\t Set timeout to n, default: 2.\n");  return 0;  }  void \*connect\_scan(void \*args)  {  scan\_args \*p = (scan\_args \*) args;  p->result = do\_connect\_scan(p->addr, p->tgt\_port);  //printf("%d ", p->tgt\_port);  return NULL;  }  void set\_timeout(int sec)  {  tout.tv\_sec = sec;  tout.tv\_usec = 0;  }  int do\_connect\_scan(in\_addr\_t addr, int port)  {  int fd, res, valopt, retflag = 1;  long arg;  unsigned int temp;  fd\_set set;  struct sockaddr\_in tgt\_addr;  fd = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  tgt\_addr.sin\_addr.s\_addr = addr;  tgt\_addr.sin\_family = AF\_INET;  tgt\_addr.sin\_port = htons(port);    arg = fcntl(fd, F\_GETFL, NULL);  arg |= O\_NONBLOCK;  fcntl(fd, F\_SETFL, arg);    // Trying to connect with timeout  res = connect(fd, (struct sockaddr \*)&tgt\_addr, sizeof(tgt\_addr));  if (res < 0) {  if (errno == EINPROGRESS) {  //fprintf(stderr, "EINPROGRESS in connect() - selecting\n");  while (1)  {  FD\_ZERO(&set);  FD\_SET(fd, &set);  res = select(fd + 1, NULL, &set, NULL, &tout);  if (res < 0 && errno != EINTR)  {  //fprintf(stderr, "Error connecting %d - %s\n", errno, strerror(errno));  retflag = 0;  goto ret;  }  else if (res > 0)  {  // Socket selected for write  temp = sizeof(int);  if (getsockopt(fd, SOL\_SOCKET, SO\_ERROR, (void\*)(&valopt), &temp) < 0)  {  //fprintf(stderr, "Error in getsockopt() %d - %s\n", errno, strerror(errno));  retflag = 0;  goto ret;  }  // Check the value returned...  if (valopt)  {  //fprintf(stderr, "Error in delayed connection() %d - %s\n", valopt, strerror(valopt));  retflag = 0;  goto ret;  }  break;  }  else  {  //fprintf(stderr, "Timeout in select() - Cancelling!\n");  retflag = 0;  goto ret;  }  }  }  else  {  //fprintf(stderr, "Error connecting %d - %s\n", errno, strerror(errno));  retflag = 0;  goto ret;  }  }  ret:  close(fd);  return retflag;  } |