자유 주제 연구
- 네트워킹

사이버보안학과 학술소학회 

송영훈
2017.11.15

계획

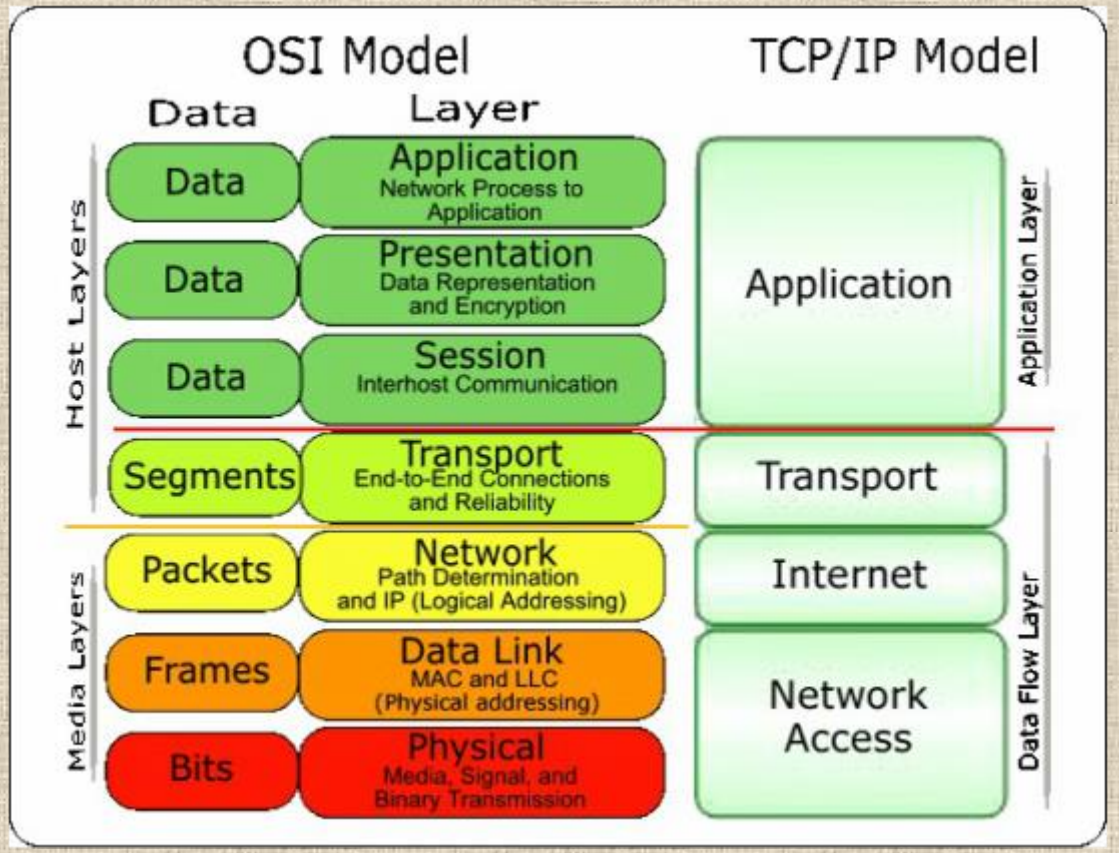
White Hacker of Information Security
WhoIs

- 1. 네트워크 기본개념, 간단한 서버 제작
- 2. 스니핑
- 3. 서비스 거부(DoS)
- 4. TCP/IP 하이재킹
- 5. 만든 서버 Exploit

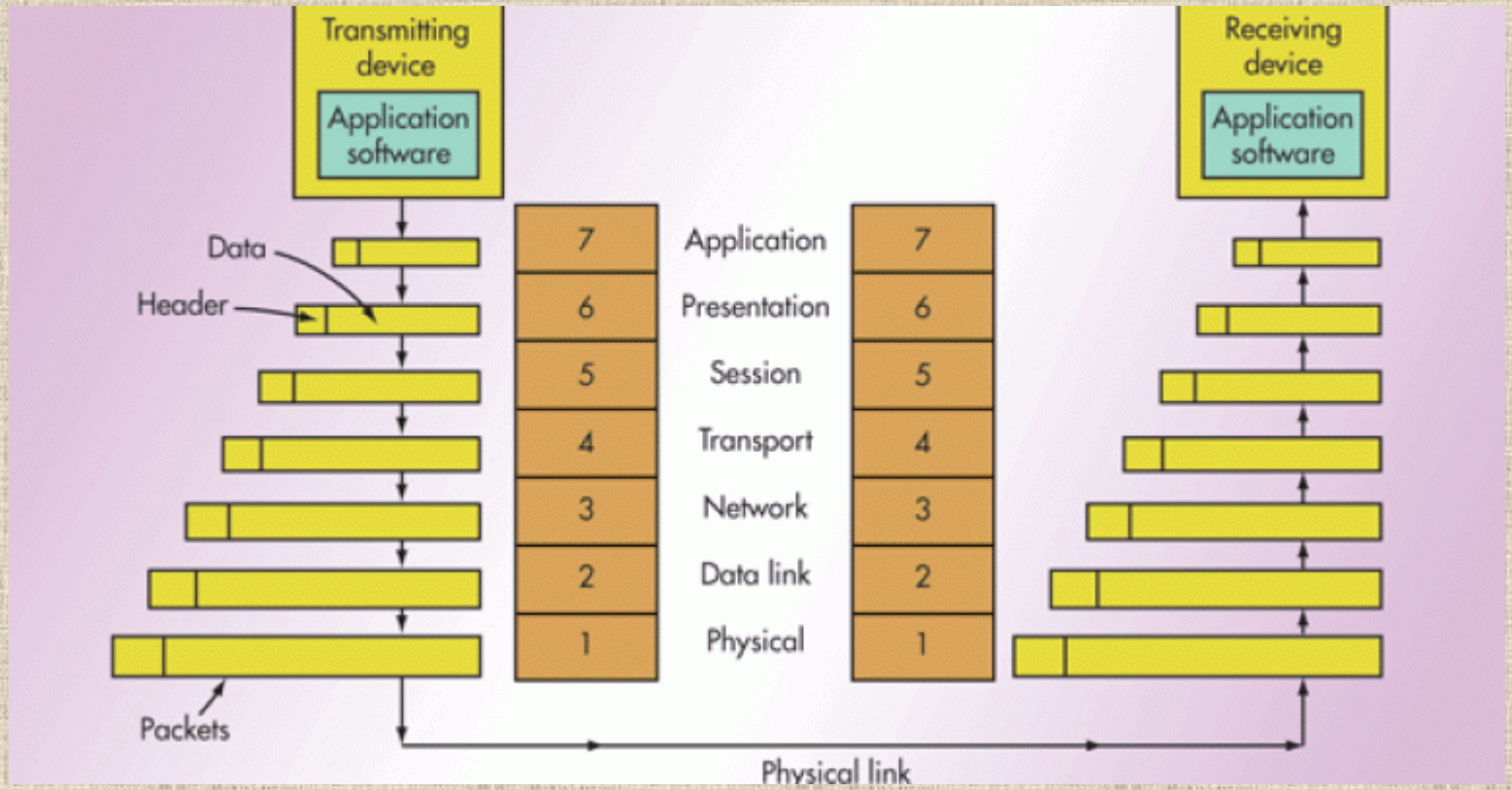
CONTENTS

네트워크 계층 구조

- OSI vs TCP/IP



데이터 전달 과정

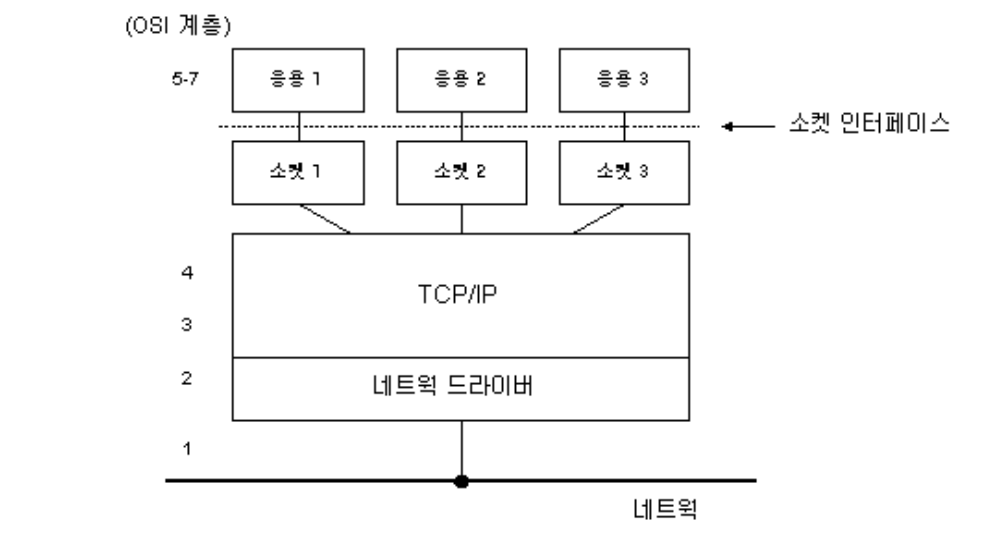

Transmitting device
Application software
Receiving device
Application software
Data
Header
7
6
5
4
3
2
1
Application
Presentation
Session
Transport
Network
Data link
Physical
7
6
5
4
3
2
1
Packets
Physical link

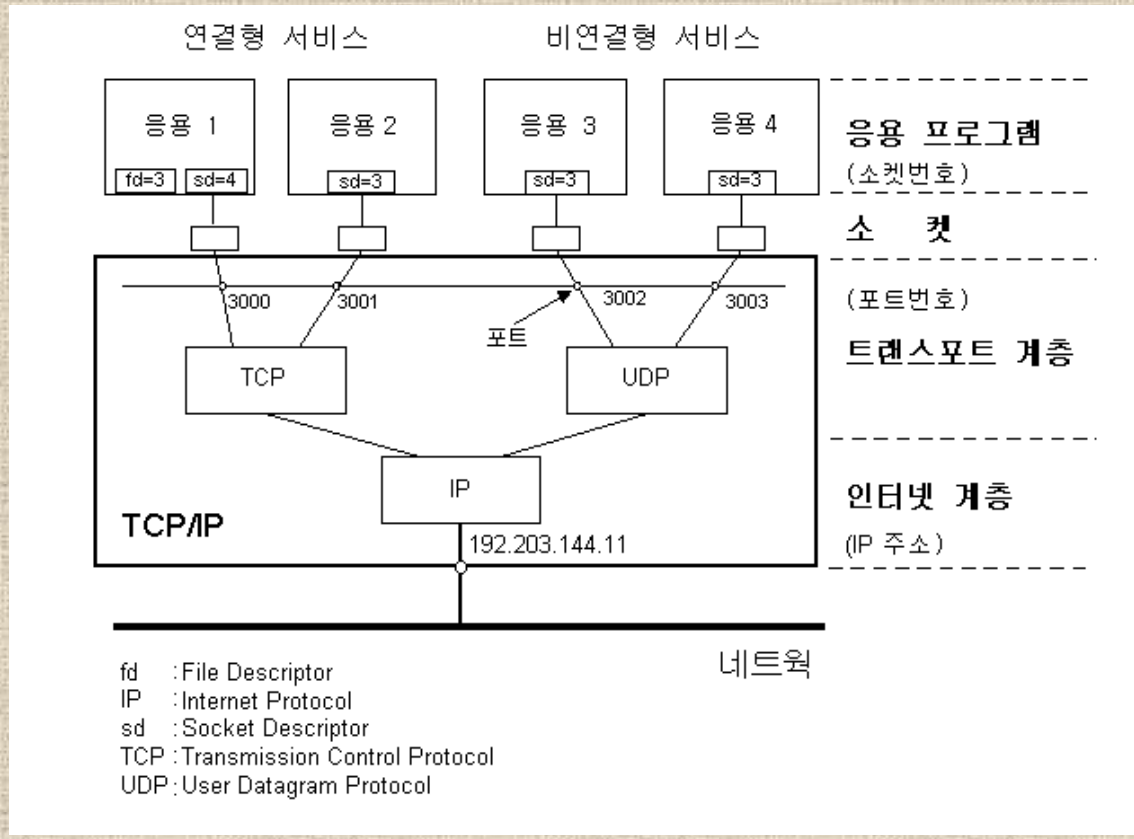
소켓

- 소켓?

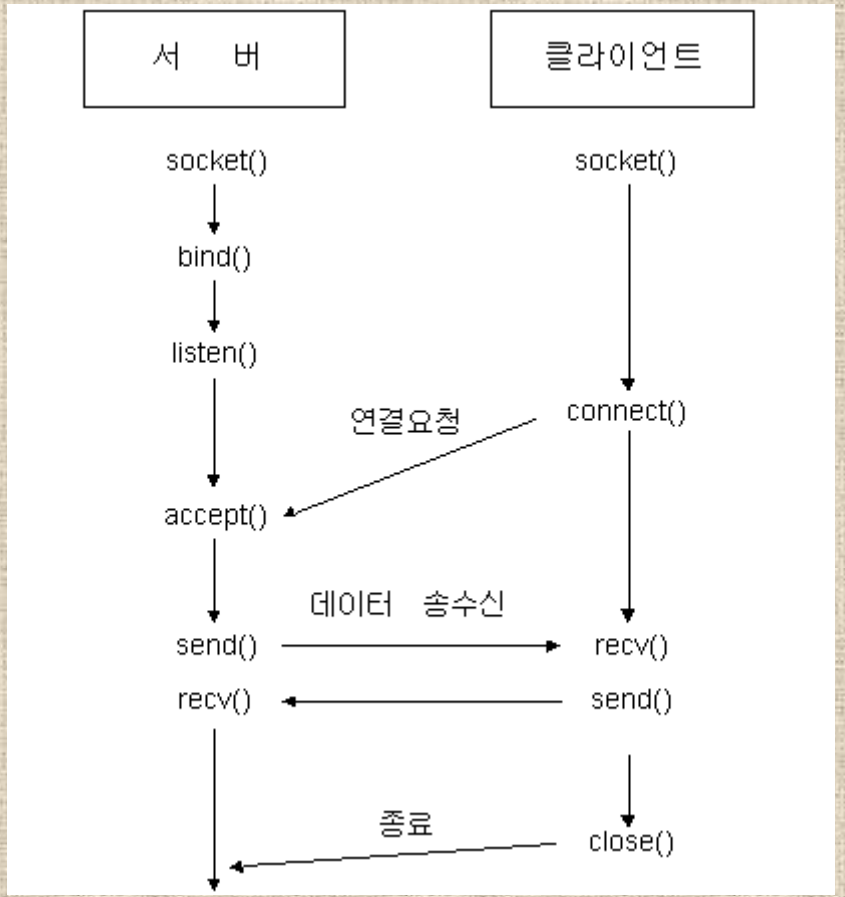
  네트워크로 데이터를 주고 받는 데 사용하는 도구.

- 양방향 통신을 제공





소켓 프로그래밍



- Socket() : 소켓 생성
- Bind() : local host와 묶음
- Listen() : 들어오는 연결을 듣고, 연결 요청을 최대 백로그 큐 사이즈 만큼 넣는다.
- Connect() : Remote host와 연결
- Accept() : 바인딩된 소켓에 들어오는 연결을 받아들임. 연결된 소켓을 식별할 수 있게 새 소켓 파일 서술자를 리턴한다.

패킷 데이터 출력 헤더 제작

White Hacker of Information Security WhoIs

- 서버 외에도 자주 쓸 예정

```
void dump(const unsigned char *data_buffer, const unsigned int length) {
   unsigned char byte;
   unsigned int i, j;
   for(i=0; i < length; i++) {
      byte = data_buffer[i];
      printf("%02x ", data_buffer[i]);  // display byte in hex
      if(((i%16)==15) || (i==length-1)) {
         for(j=0; j < 15-(i%16); j++)
            printf("   ");
         printf("| ");
         for(j=(i-(i%16)); j <= i; j++) {  // display printable bytes from line
            byte = data_buffer[j];
            if((byte > 31) && (byte < 127)) // outside printable char range
               printf("%c", byte);
            else
               printf(".");
         }
         printf("\n"); // end of the dump line (each line 16 bytes)
      } // end if
   } // end for
}
```

Simple Server 제작

White Hacker of Information Security
WhoIs

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "hacking.h"

#define PORT 7890 // the port users will be connecting to

int main(void) {
   int sockfd, new_sockfd;  // listen on sock_fd, new connection on new_fd
   struct sockaddr_in host_addr, client_addr;   // my address information
   socklen_t sin_size;
   int recv_length=1, yes=1;
   char buffer[1024];

   if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
      fatal("in socket");

   if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
      fatal("setting socket option SO_REUSEADDR");

   host_addr.sin_family = AF_INET;      // host byte order
   host_addr.sin_port = htons(PORT);    // short, network byte order
   host_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
   memset(&(host_addr.sin_zero), '\0', 8); // zero the rest of the struct
```

Simple Server 제작

```
if (bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct sockaddr)) == -1)
   fatal("binding to socket");

if (listen(sockfd, 5) == -1)
   fatal("listening on socket");

while(1) {    // Accept loop
   sin_size = sizeof(struct sockaddr_in);
   new_sockfd = accept(sockfd, (struct sockaddr *)&client_addr, &sin_size);
   if(new_sockfd == -1)
      fatal("accepting connection");
   printf("server: got connection from %s port %d\n",
          inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
   send(new_sockfd, "Hello World!\n", 13, 0);
   recv_length = recv(new_sockfd, &buffer, 1024, 0);
   while(recv_length > 0) {
      printf("RECV: %d bytes\n", recv_length);
      dump(buffer, recv_length);
      recv_length = recv(new_sockfd, &buffer, 1024, 0);
   }
   close(new_sockfd);
}
return 0;
```

Simple Server Test

```
server: got connection from 192.168.54.100 port 57612
RECV: 5 bytes
48 69 7e 0d 0a                                  | Hi~..
RECV: 16 bytes
54 68 69 73 20 69 73 20 54 65 73 74 2d 2d 0d 0a | This is Test--..
RECV: 6 bytes
47 6f 6f 64 0d 0a                               | Good..
RECV: 2 bytes
0d 0a                                           | ..
```

<< 7890번 포트로 열린 서버

텔넷으로 접속>>

```
root@server:~# telnet 192.168.54.129 7890
Trying 192.168.54.129...
Connected to 192.168.54.129.
Escape character is '^]'.
Hello World!
Hi~
This is Test--
Good
```

TCP 간략 소개

White Hacker of Information Security WhoIs

- TCP(Transmission Control Protocol) : 인터넷에 연결된 컴퓨터에서 실행되는 프로그램 간에 일련의 옥텟을 안정적으로, 순서대로, 에러없이 교환할 수 있게 한다. (전송 계층)

- 연결설정 - Three-way Handshake

  1) 상대에게 통신을 하고 싶다는 메시지를 보낸다. (SYN)

  2) 상대는 그 메시지에 대한 응답 +
     나도 통신 준비가 되었다는 메시지를 보낸다. (SYN-ACK)

  3) 2번에서 받은 메시지에 응답을 보낸다. (ACK)

HTTP 간략 소개

- HTTP(Hyper Text Transfer Protocol) :

클라이언트와 서버 사이에 이루어지는 응답/요청 프로토콜.
주로 HTML 문서를 주고받는데 사용, TCP, 80번 포트 사용 (응용 계층)

- 클라이언트가 다음과 같은 메시지
  ex) GET /images/sample.gif

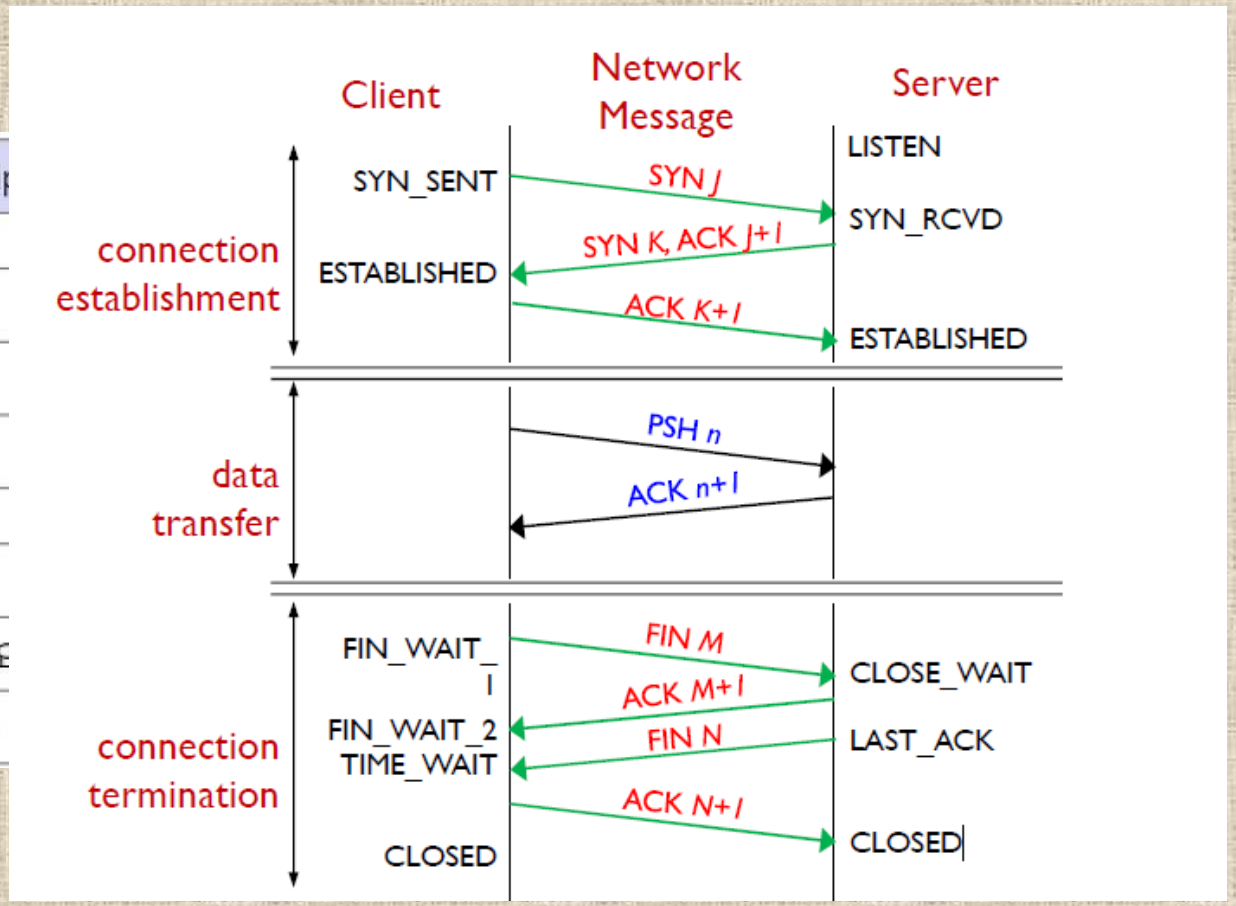
- 서버는 다음과 같이 세 자리수로
  ex) HTTP/1.1 200 OK …

상태코드	의미	범위 별 의미
200	클라이언트의 요청이 성공적으로 끝남	클라이언트 요청 성공
400	요청한 method를 지원하지 않음	클라이언트 요청에 대한 방향 재정의, 추가 동작 필요
401	인증오류	
403	사용자 허가 모드 오류	
404	요청한 파일이 존재하지 않음	
405	요청한 method를 지원하지 않음	
500	Internal Server Error(실행오류)	서버오류

HTTP 간략 소개

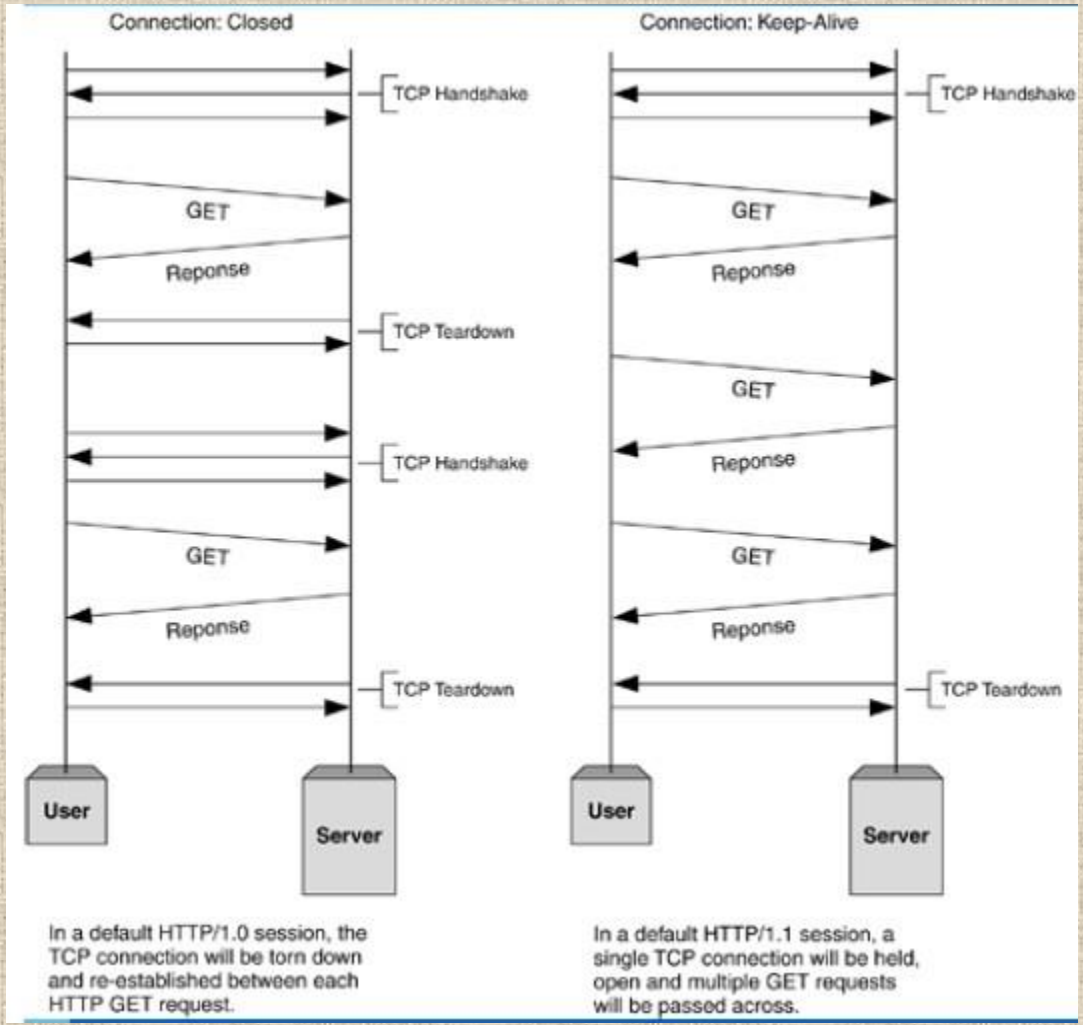
- 주요 메소드 및 연결 과정

Method	Descrip
GET	자원 요청
POST	Entity를 포함한 자원 요청
HEAD	HTTP Header 정보만 수신
TRACE	Request의 루프백 테스트
PUT	URL에 자원을 생성
DELETE	URL의 자원을 삭제
OPTIONS	응답 가능한 HTTP 메소드를 요
CONNECT	터널링의 목적으로 연결 요청



HTTP 간략 소개

- HTTP 1.0 vs 1.1



간단한 웹 서버 제작

White Hacker of Information Security
WhoIs

```c
int get_file_size(int fd) {
   struct stat stat_struct;

   if(fstat(fd, &stat_struct) == -1)
      return -1;
   return (int) stat_struct.st_size;
}
```

```c
#define PORT 80   // the port users will be connecting to
#define WEBROOT "./webroot" // the web server's root directory

void handle_connection(int, struct sockaddr_in *); // handle web requests
int get_file_size(int); // returns the filesize of open file descriptor

int main(void) {
   int sockfd, new_sockfd, yes=1;
   struct sockaddr_in host_addr, client_addr;   // my address information
   socklen_t sin_size;

   printf("Accepting web requests on port %d\n", PORT);

   if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
```

```c
while(1) {   // Accept loop
   sin_size = sizeof(struct sockaddr_in);
   new_sockfd = accept(sockfd, (struct sockaddr *)&client_addr, &sin_size);
   if(new_sockfd == -1)
      fatal("accepting connection");

   handle_connection(new_sockfd, &client_addr);
}
return 0;
```

```c
EUSEADDR, &yes, sizeof(int)) == -1)
EADDR");

 host byte order
 short, network byte order
 // automatically fill with my IP
 // zero the rest of the struct

 if (bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct sockaddr)) == -1)
    fatal("binding to socket");

 if (listen(sockfd, 20) == -1)
    fatal("listening on socket");
```

간단한 웹 서버 제작

White Hacker of Information Security
WhoIs

```
void handle_connection(int sockfd, struct sockaddr_in *client_addr_ptr) {
   unsigned char *ptr, request[500], resource[500];
   int fd, length;

   length = recv_line(sockfd, request);

   printf("Got request from %s:%d \"%s\"\n",
         inet_ntoa(client_addr_ptr->sin_addr), ntohs(client_addr_ptr->sin_port), request);

   ptr = strstr(request, " HTTP/"); // search for valid looking request
   if(ptr == NULL) { // then this isn't valid HTTP
      printf(" NOT HTTP!\n");
   } else {
      *ptr = 0; // terminate the buffer at the end of the URL
      ptr = NULL; // set ptr to NULL (used to flag for an invalid request)
      if(strncmp(request, "GET ", 4) == 0)  // get request
         ptr = request+4; // ptr is the URL
      if(strncmp(request, "HEAD ", 5) == 0) // head request
         ptr = request+5; // ptr is the URL

      if(ptr == NULL) { // then this is not a recognized request
         printf("\tUNKNOWN REQUEST!\n");
      } else { // valid request, with ptr pointing to the resource name
         if (ptr[strlen(ptr) - 1] == '/')  // for resources ending with '/'
            strcat(ptr, "index.html");     // add 'index.html' to the end
         strcpy(resource, WEBROOT);     // begin resource with web root path
         strcat(resource, ptr);         //  and join it with resource path
         fd = open(resource, O_RDONLY, 0); // try to open the file
         printf("\tOpening \'%s\'\t", resource);
```

간단한 웹 서버 제작

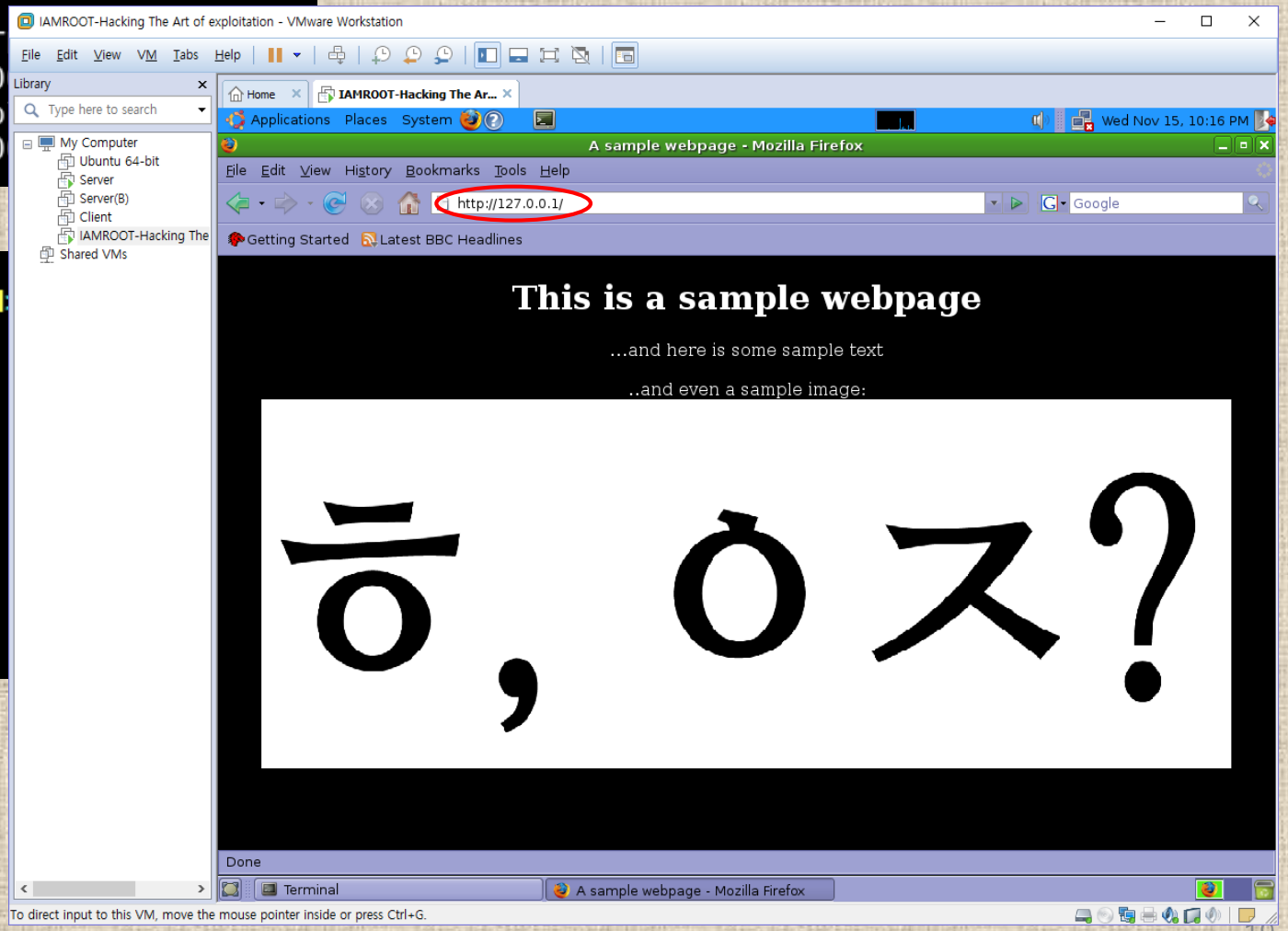
White Hacker of Information Security
WhoIs

```
        if(fd == -1) { // if file is not found
            printf(" 404 Not Found\n");
            send_string(sockfd, "HTTP/1.0 404 NOT FOUND\r\n");
            send_string(sockfd, "Server: Tiny webserver\r\n\r\n");
            send_string(sockfd, "<html><head><title>404 Not Found</title></head>");
            send_string(sockfd, "<body><h1>URL not found</h1></body></html>\r\n");
        } else {      // otherwise, serve up the file
            printf(" 200 OK\n");
            send_string(sockfd, "HTTP/1.0 200 OK\r\n");
            send_string(sockfd, "Server: Tiny webserver\r\n\r\n");
            if(ptr == request + 4) { // then this is a GET request
                if( (length = get_file_size(fd)) == -1)
                    fatal("getting resource file size");
                if( (ptr = (unsigned char *) malloc(length)) == NULL)
                    fatal("allocating memory for reading resource");
                read(fd, ptr, length); // read the file into memory
                send(sockfd, ptr, length, 0);  // send it to socket
                free(ptr); // free file memory
            }
            close(fd); // close the file
        } // end if block for file found/not found
    } // end if block for valid request
} // end if block for valid HTTP
shutdown(sockfd, SHUT_RDWR); // close the socket gracefully
```

웹 서버 제작

```
Accepting web requests on port 80
Got request from 127.0.0.1:40606 "GET / HTT
        Opening './webroot/index.html'    20
Got request from 127.0.0.1:40607 "GET /samp
        Opening './webroot/sample.png'    20
```

```
<html>
<head><title>A sample webpage</title></head
<body bgcolor="#000000" text="#ffffffff">
<center>
<h1>This is a sample webpage</h1>
...and here is some sample text<br>
<br>
..and even a sample image:<br>
<img src="sample.png"><br>
</center>
</body>
</html>
```



Reference

- 주 사용 언어 : C 언어
- Server OS : Ubuntu 7.04 – iamroot 커뮤니티 제공
- Client OS : Ubuntu 16.04
- 이미지 제공 : WhoIs

- Iamroot 사이트 : http://www.iamroot.org/xe/

Q & A

White Hacker of Information Security
WhoIs