C로 알아보는
UDP 통신

이현환(NOON)
haonun@gmail.com
http://noon.tistory.com

Hacking Study Grup E.Y.E

목차

서론

본론

결론

서론

1. 이번에는 왜하는 걸까..?

저번 TCP/IP 소켓 프로그래밍에서 중요성을 강조했듯이 현재 IT 산업에서 중요한 부분을 맡고 있는게 네트워크 부분입니다. 서로간의 통신으로 좀더 유연한 어플리케이션을 구현할 수 있고 또한 사용자에게 좀다 빠르고 나은 환경을 제공합니다. 저희 팀에서도 추후 여러가지 기술과 , 툴제작등 여러행동을 할때에도 이 네트워크 부분은 빠지지 않습니다. 이중 이번에는 TCP/IP와 함께 많이 쓰이는 통신 프로토콜인 UDP 에 대해 알아보고 UDP가 어떤곳에 사용되는지 , 실제 코드로는 어떻게 사용되는지 어떤식으로 해킹,보안에 응용할 수 있는지 보겠습니다.

본론

2. UDP란?

-UDP 정의-
UDP[유디피]는 IP를 사용하는 네트웍 내에서 **컴퓨터들 간에 메시지들이 교환될 때 제한된 서비스만을 제공하는 통신 프로토콜**이다. UDP는 TCP의 대안이며, IP와 함께 쓰일 때에는 **UDP/IP**라고 표현하기도 한다. TCP와 마찬가지로 UDP도 한 컴퓨터에서 다른 컴퓨터로 **데이터그램**이라고 불리는 실제 데이터 단위를 받기 위해 IP를 사용한다. 그러나 **UDP는 TCP와는 달리, 메시지를 패킷(데이터그램)으로 나누고, 반대편에서 재조립하는 등의 서비스는 제공하지 않으며, 특히 도착하는 데이터 패킷들의 순서를 제공하지 않는다.** 이 말은 UDP를 사용하는 응용프로그램은, 전체 메시지가 올바른 순서로 도착했는지에 대해 확인할 수 있어야한다는 것을 의미한다. 교환해야할 데이터가 매우 적은(그러므로 재조립해야할 메시지도 매우 적은) 네트웍 응용프로그램들은 처리시간 단축을 위해 TCP 보다 UDP를 더 좋아할 수 있다. 일례로 TFTP는 TCP 대신에 UDP를 사용한다.

UDP는 IP 계층에서 제공되지 않는 두 개의 서비스를 제공하는데, 하나는 다른 사용자 요청을 구분하기 위한 포트 번호와, 도착한 데이터의 손상여부를 확인하기 위한 체크섬 기능이다.
-팀즈-

저번에 말씀드렸던 내용입니다. **TCP는 연결형 , 신뢰성을 기반으로한 전송 프로토콜**입니다. 서버가 Listen을 통해 기다리고 클라이언트가 접속신호를 주고 확인을 받고 해서 통신이 되었지요. 그러나 UDP는 조금 다릅니다. 간단히 설명하자면 서로를 믿지 않습니다. 그저 데이터 전송과 수신만이 존재합니다.

OSI 통신 모델에서, UDP는 TCP와 마찬가지로 4계층인 트랜스포트 계층에 속합니다. 즉 UDP와 TCP는 동급의 프로토콜로 데이타를 전송하기 위해서 사용되는 프로토콜입니다.

UDP(User Datagram Protocal)는 비연결이고, 신뢰성이 없는 전송 프로토콜 입니다.
UDP는 호스트 간 통신 대신에 프로세스 간 통신을 제공하는 것을 제외하고는 IP 서비스에 어떠한 것도 추가하지 않습니다. 또한 매우 제한적인 오류검사를 한다. 최소한의 오버헤드를 가진 간단한 프로토콜이라는 장점을 가집니다. 그래서 서버나 클라이언트에서 처리해야할 일의 수가 줄어들게 되고 자연스럽게 통신 속도가 향상됩니다.

TCP는 서로 통신을 하기전에 상대방을 확인하는 절차를 가짐으로써, session을 맺고 연결된 session 을 통해서 데이타의 흐름이 이루어집니다.
그러나 UDP 는 session을 만들지 않고 그저 데이터의 전송과 수신만을 합니다. 그래서 사용자가 UDP 서비스를 하는 서버로 메시지를 보냈다고 하더라도, 메시지가 실제로 도착되었는지 알수가 없습니다. 메시지가 전송이 됫을수도 있고 안됫을수도 있는거죠.

그래서 TCP와 달리 신뢰할수가 없다. TCP는 프로토콜자체에 메시지가 제대로 보내졌음을 체크할수 있는 다양한 장치를 가지고 있습니다. 즉 각 패킷에 순서를 매겨서, 순서가 뒤엉키지 않도록 재조립하며, 일정시간 동안 패킷이 도착하지 않으면, 해당 패킷을 다시 보내달라고 요청할수도 있습니다. 그러나 UDP는 이러한 어떠한 장치를 가지고 있지 않습니다. UDP 로 전송된 패킷은 순서가 뒤바뀔수도 있으며, 중간에 패킷이 손실될수도 있습니다. 프로토콜 차원에서 패킷의 순서가 뒤바뀌었는지, 패킷이 손실되었는지 알수 있는 방법은 없습니다.

UDP 패킷에 신뢰성을 주기 위해서는 프로그램에서 직접 코딩을 해주어야만 합니다.
패킷을 보낼 때 패킷에다가 일련번호 , 체크 , 확인 등 여러 가지 상태를 서버가 파악할수 있도록 패킷을 전송하고 또한 재전송시에도 지금 상태가 어떻한지 알려줄수 있는 무언가가 필요합니다. 예를 들자면 클라이언트에서 서버로 데이터를 전송시 UDP에서는 전송이 됫는지 서버가 꺼져있는지 켜져있는지 확인이 불가능합니다. 이를 방지하기 위해 클라이언트가 패킷전송시 맨앞에 1이라는 데이터를 같이 전송하고 서버가 받으면 맨앞에 0이라는 데이터를 바로 클라이언트에게 보내고 클라이언트가 4초이내에 0이라는 데이터를 받지 못하면 서버가 꺼져있다. 이런식으로 판단할 수도 있습니다.

이렇듯 UDP 는 단순히 데이타 그램 위주의 통신을 하기 때문에, 데이타 그램 지향 프로토콜 이라고 불리우기도 합니다. 실제로 UDP는 User Datagram Protocol 의 줄임말입니다.

3. UDP 의 활용

UDP에 대해 약간 알아보았습니다. 안정성보다는 속도에 치중된 프로토콜이기에 좀더 빠른 서비스에 적합합니다.

일단 음성및 비디오를 위한 실시간 스트리밍 서비스에 사용됩니다. 음성서비스를 TCP로 할 경우 중간에 패킷이 손상가거나 잃어버리면 서비스를 중단하고 다시 연결요청을 하고 재접속을 해야하기에 통화중 전화가 끊어진다거나 하는 현상이 발생합니다. 이건 마치 우리가 전화를 할때 중간에 약간의 잡음이 생겼다고 해서, 전화가 중단되는 것과 마찬가지의 상황입니다. 우리는 약간의 잡음때문에 (혹은 한두자 정도 언어가 전달이 안되는) 그걸

교정하느라고 서비스가 중단되는 것 보다는 서비스질이 약간 떨어지더라도 계속적으로 서비스가 되는걸 원할것입니다. 그래서 UDP를 사용하면면 패킷이 도착했는지 체크따위 하지 않으므로 만약 패킷을 잃어버렸다 하여도 모르고 계속 진행하게 됩니다. 즉 통신품질보다는 통신의 연속성이 더욱 중요시 되는곳에 유용하게 사용될수 있다. (물론 TCP로도 구현할수 있으며, 상당수의 서비스가 TCP로 서비스 된다. 다만 이러한 특징을 가지고 있습니다.

또한 스타크래프트에서도 사용됩니다. start craft 의 베틀넷 서비스가 아닐가 싶다. 이 베틀넷 서비스에는 수많은 유저가 접속해서 사용할건데, 서비스의 모든 부분에 TCP를 사용하

기에는 TCP는 너무 느린 감이 있다. 특히 게임을 할때 서로 교환되는 수많은 패킷의 경우 매우 중요한 데이터가 아니므로, 그리고 게임의 흐름이 끊기면 안되므로 UDP로 처리되는게 더 유리 할 것입니다. 멀티플레이에서도 UDP모드를 따로 지원하기도 합니다.

이외에도
DNS (Domain Name Service)
NFS (Network File System)
SNMP (Simple Network Management Protocol)
Trivial FTP (TFTP)
Bootstrap Protocol (BOOTP)
Dynamic Host Configuration Protocol (DHCP)
Routing Information Protocol (RIP)
에도 사용됩니다.

4. C로 알아보는 UDP 통신

간단하게 ECHO 서버와 클라이언트로 UDP통신을 어떻게 사용하는지에 대해 알아보겠습니다.

echoServer.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#define MAXLINE 511
int main(int argc , char* argv[] )
{       struct sockaddr_in servaddr, cliaddr;
        int s, nbyte , addrlen = sizeof(struct sockaddr);
        char buf[MAXLINE];
        if(argc != 2 )
        {
        printf("usage port");
        exit(0);
        }
        if((s = socket(AF_INET , SOCK_DGRAM, 0 )) < 0)
                        {

                                perror("socket error");
                                exit(0);
                        }
        bzero((char*)&servaddr, addrlen);
        bzero((char*)&cliaddr, addrlen);
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
        servaddr.sin_port = htons(atoi(argv[1]));
        if(bind(s,(struct sockaddr*)&servaddr , addrlen ) <0)
        {
        perror("bind Error");
        exit(0);
        }
```

```
                while(1)
        {
        puts("Server : Waiting Request₩n");
        nbyte = recvfrom(s,buf,MAXLINE,0,(struct sockaddr*)&cliaddr, &addrlen);
        if(nbyte<0)
        {
        perror("recv error");
        exit(0);
        }
        buf[nbyte] = 0;
                printf("%d byte Recv : %s₩n",nbyte,buf);
        if(sendto(s,buf,nbyte,0,(struct sockaddr*)&cliaddr, addrlen) < 0 )
        {
        perror("send Error");
        exit(0);
        }
        puts("send Complete₩n");
        }
        return 0;
}
```

echoClinet

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#define MAXLINE 511
int main(int argc , char* argv[])
{struct sockaddr_in servaddr;
int s , nbyte , addrlen = sizeof(servaddr);
char buf[MAXLINE+1];
if(argc != 3 )
{printf("usage : ip , port");
exit(0);
```

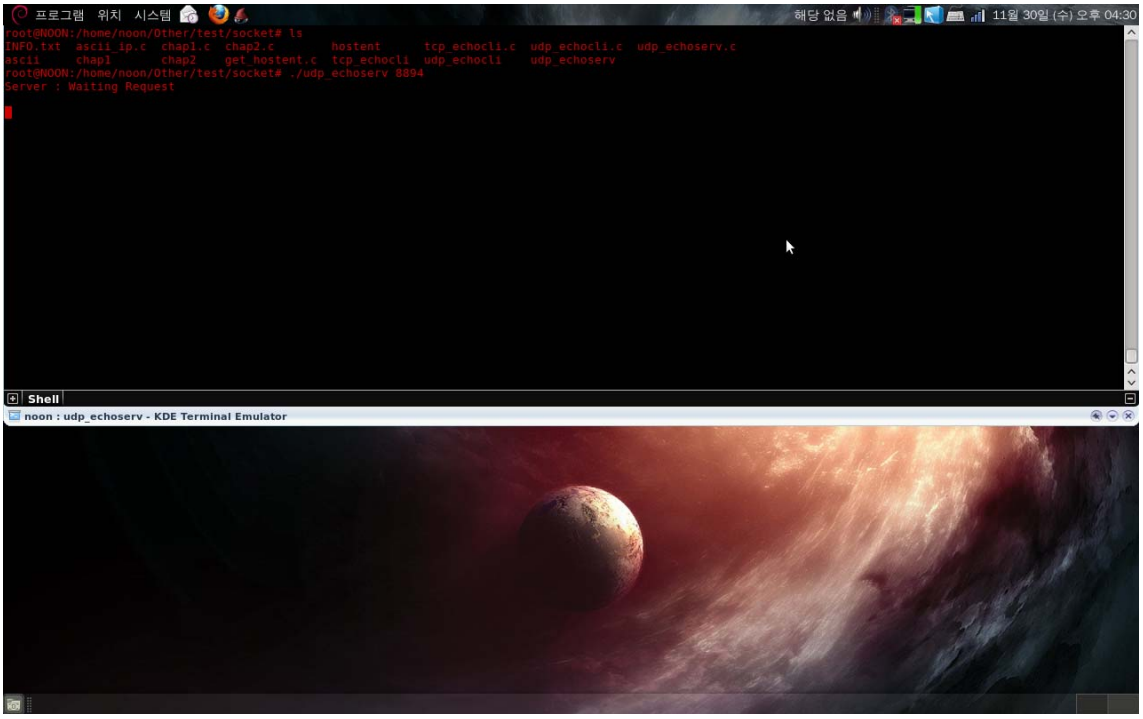
```
}if((s = socket(PF_INET , SOCK_DGRAM , 0)) < 0)
{perror("socket fail");
exit(0);
}bzero((char*)&servaddr ,sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr(argv[1]);
servaddr.sin_port = htons(atoi(argv[2]));
if(fgets(buf,MAXLINE,stdin) == NULL)
{printf("fget error");
exit(0);
}if(sendto(s,buf,strlen(buf),0,(struct sockaddr*)&servaddr, addrlen) < 0)
{perror("send error");
exit(0);
}if((nbyte = recvfrom(s,buf,MAXLINE,0,(struct sockaddr*)&servaddr , &addrlen
                                        ))  < 0)
{       perror("recv error");
        exit(0);
}buf[nbyte] = 0;
printf("%s₩n",buf);
close(s);
return 0;
}
```

위 프로그램 작성후 테스트를 위해 사용될 PC입니다.

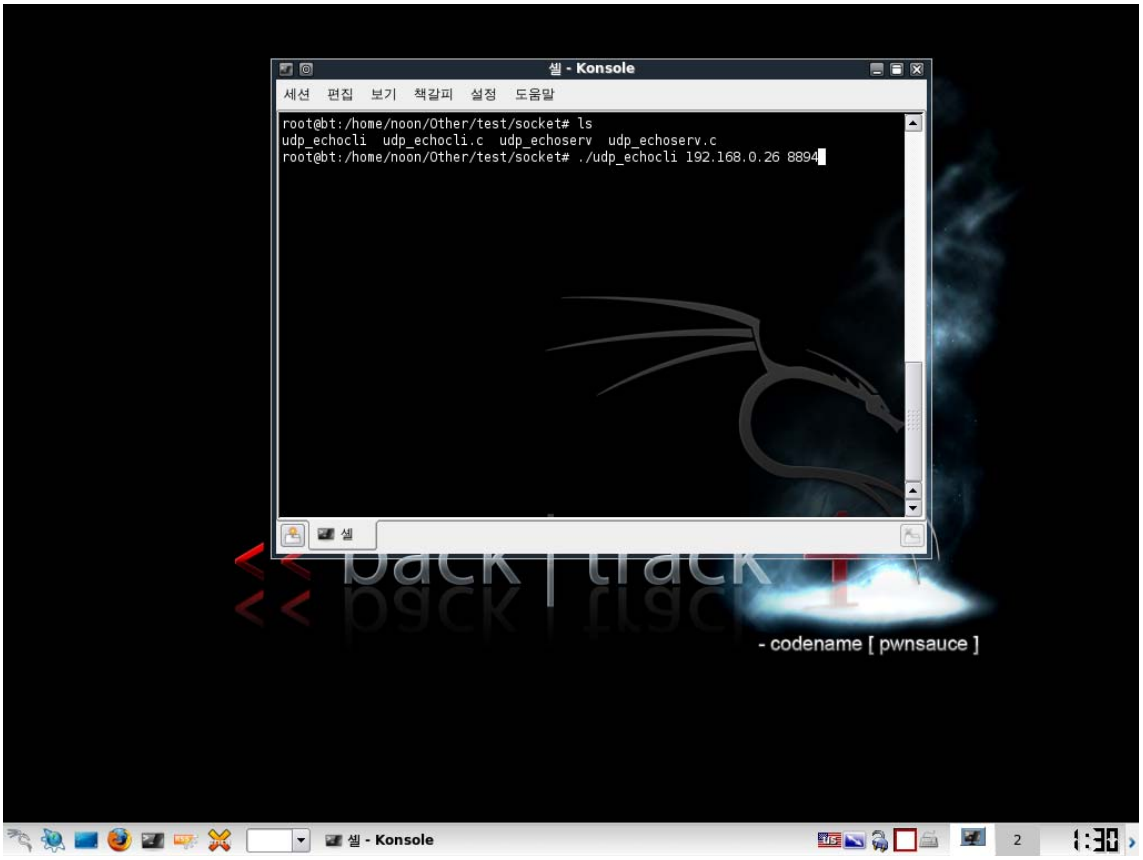
NoteBook (서버)
OS : Debian
IP : 192.168.0.26

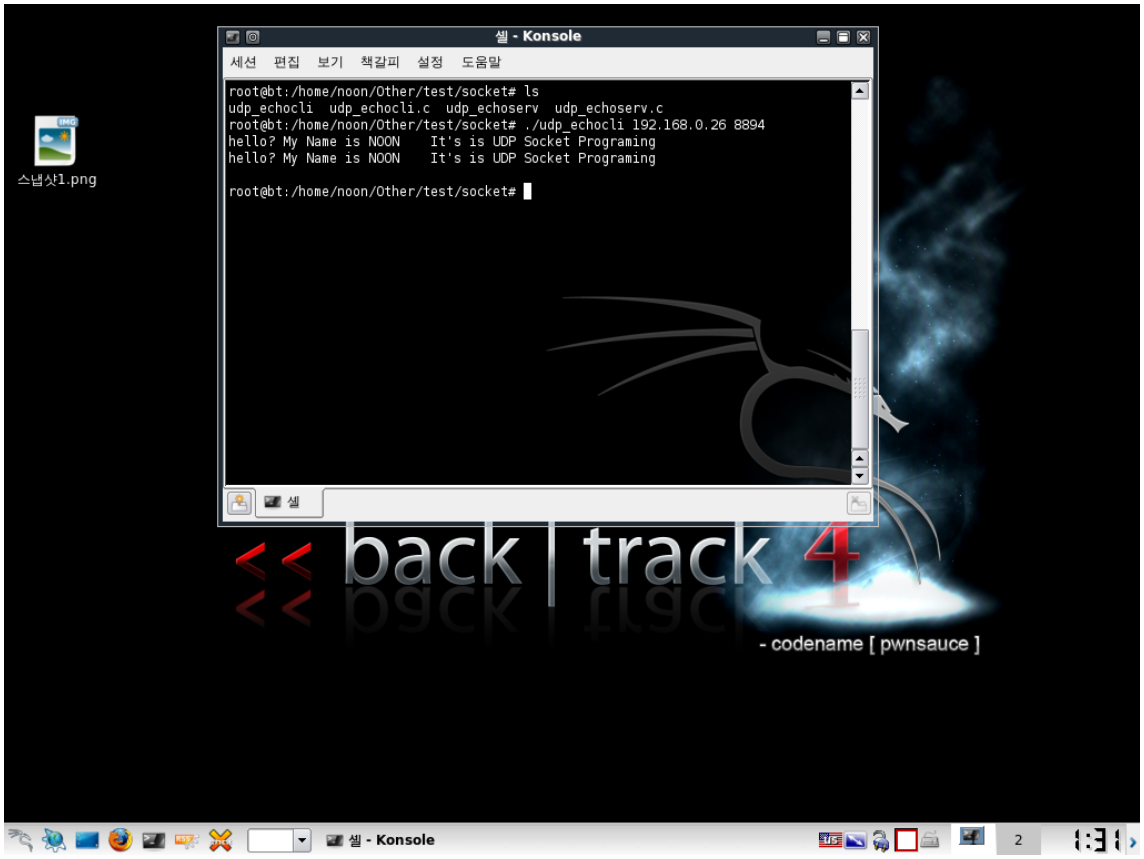
DeskTop (클라이언트)
OS : BackTrack4
IP : 192.168.0.3

노트북에서 echoserver를 실행하여 8894라는 포트를 열었습니다. 8894는 UDP를 위한 포트가 됩니다.

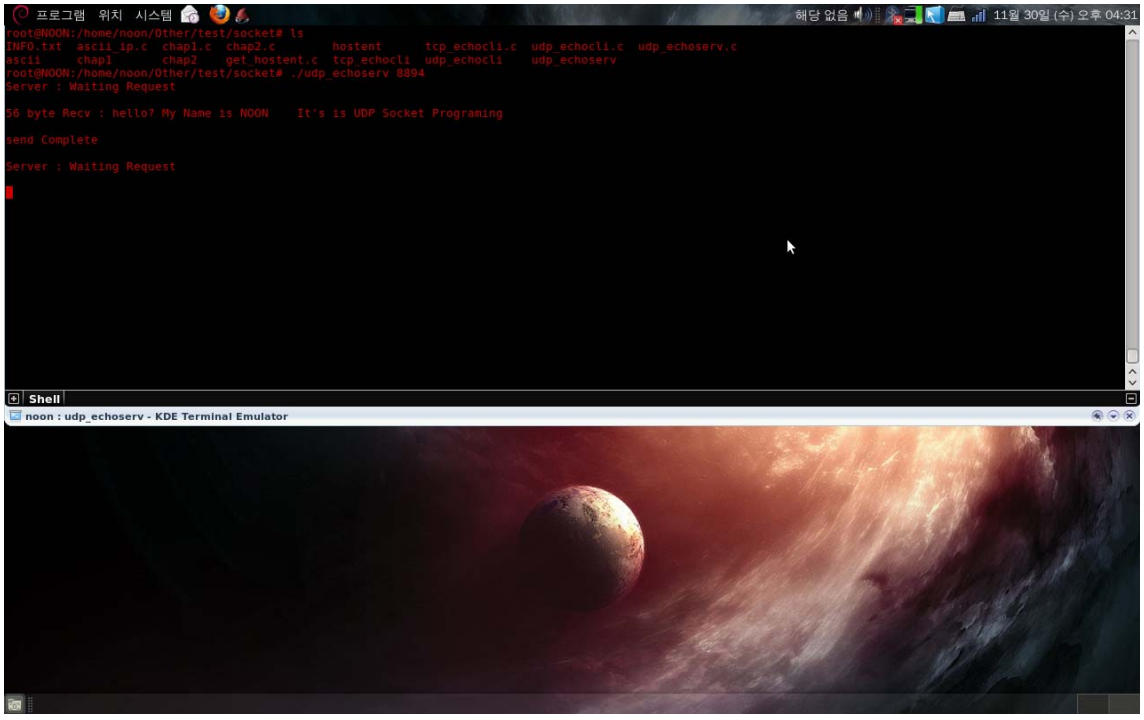


클라이언트가 접속합니다.



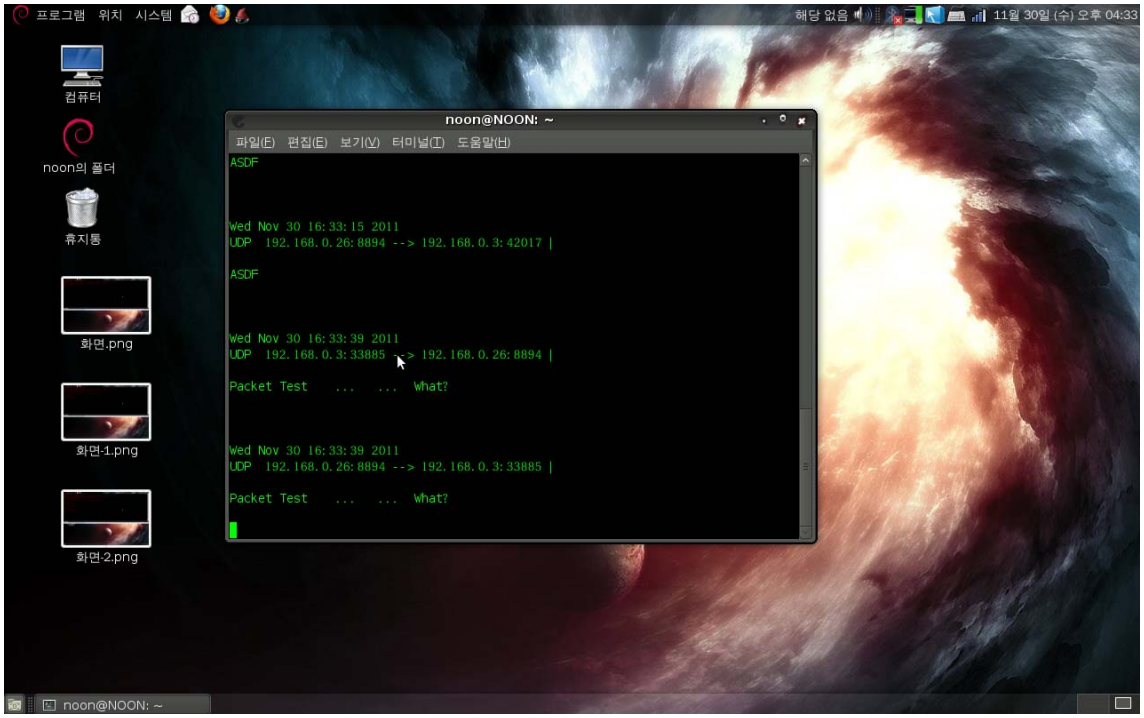


메시지를 날립니다.



서버에서 메시지를 받고 클라이언트로 보내줍니다.

위 프로그램은 이런 방식으로 구동이 되며 ettercap이나 dsniff등 패킷을 캡쳐해서 분석해 보면 다음과 같이 UDP를 이용해 데이터의 통신이 되고있을을 알수 있습니다.

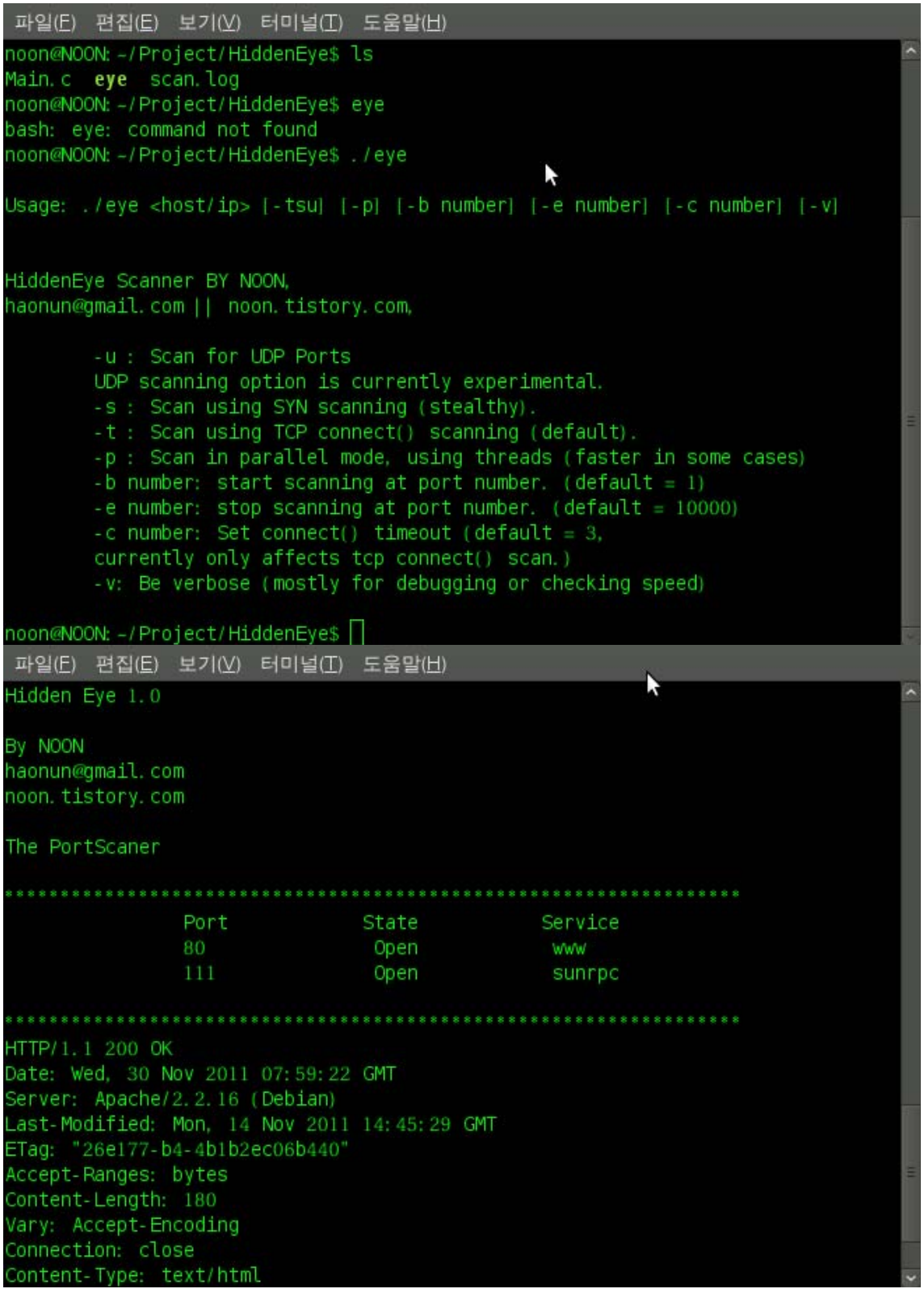


서버의 8894 포트와 클라이언트의 33885 포트를 이용해 서로 데이터를 주고 받았습니다. 여기서. 클라이언트의 포트가 왜 33885 인가 함은. 클라이언트에서 데이터를 전송시 나갈 포트를 지정해 주는데 위의 코드에서 보면 나갈 포트를 시스템에서 랜덤하게 잡아주게 설정해놧기에 임의의 포트로 전송을 시도하게 됩니다.

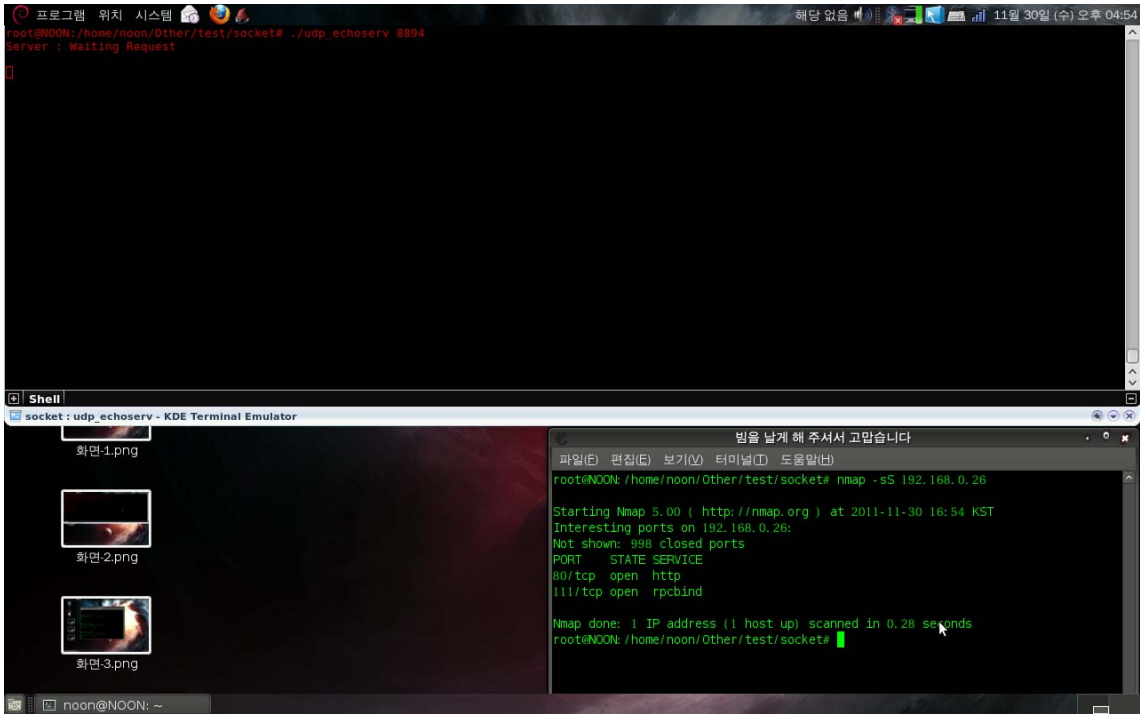
5. UDP 발전방향

많은 서비스들이 TCP를 이용해 서비스 하고 있습니다. 그래서 빈틈이 많은 서버관리자들은 네트워크를 체크시 UDP를 놓칠 수 있습니다.

현재 진행중인 HiddenEye 프로그램으로 기본옵션 으로 확인시 UDP포트에 대한 정보를 가져 오지 않습니다. 무능한 관리자라면 기본값만 확인하고 안전하다고 판단할수도 있는 상황입니다.



또한 전세계적으로 잘 알려진 NMAP을 통한 스캔에서도 UDP를 따로 체크해 주지 않는 이상 찾아낼수가 없습니다.



이처럼 사용자들에게 덜 알려진 프로토콜이나 접속 방법등은 공격자에게 매우 유용한 길이 될것이고 그로 말미암아 시스템을 장악당하게 되는 일또한 일어날 수 있습니다. 잘 찾아보시면 UDP를 이용한 공격법도 많이 존재합니다.

결론

이처럼 TCP와 함께 많이 사용되는 UDP프로토콜에 대해 알아봄으로써 좀더 유연한 프로그래밍 지식과 공격기법 , 방어기법등 여러 가지에 활용 될 수 있도록 기반을 쌓는 시간이 되셨으면 좋겠습니다. 감사합니다.

문의 : hanun@gmail.com :: http://noon.tistory.com