

UDP Remote Controls

by: Angelo Rosiello, 10/04/2004

<http://www.securitydocs.com/library/2626>

1 Introduction

I want to illustrate, with this article, the possibility to control servers with the UDP protocol. In order to exemplify the topic in question, I conceptualized the program. You can find the complete package here: <http://packetstormsecurity.org/UNIX/penetration/udp-remote-final.tar.gz>

The sources of the project, are below, point number 5.

2 UDP Basics

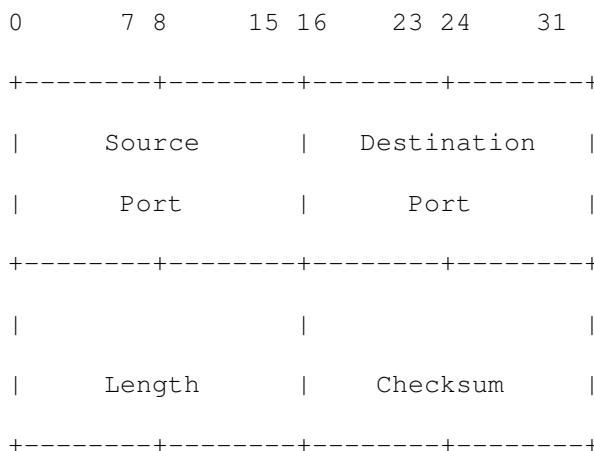
Before describing the program functions and services, I thought that it was useful to explain some important topics about the UDP protocol, that is, the basic element of the whole project.

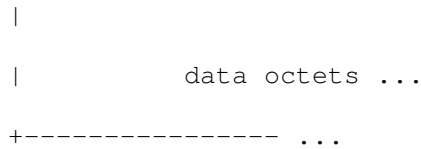
User Datagram Protocol (rfc 768), together with TCP, is the the transport layer protocol of the ISO/OSI model:

7. APPLICATION LAYER
6. PRESENTATION LAYER
5. SESSION LAYER
4. TRANSPORT LAYER (TCP-UDP) ← We are here!
3. NETWORK LAYER (IP)
2. DATA LINK LAYER
1. PHYSICAL LAYER

The generic format of the UDP datagram is the following:

Format -----





(User Datagram Header Format)

Source Port : (16 bit)	The source port is an optional field, it indicates the port from which the packet was sent and eventually toward which the answer is expected. If no value is set, the default value is forced to 0.
Destination Port : (16 bit)	It indicates the destination port of the final peer process.
Length : (32 bit)	It specifies in bytes the datagram length in its entirety (header+payload).
Checksum : (16 bit)	It develops typical functions as error-control on the TCP IU, added a header containing the source IP address, the destination IP address, the number of TCP protocol and the number (in bytes) of the UDP fragment.

The error-control procedure realized by the checksum field is optional, in order to relieve the protocol as much as possible.

3 How to spoof datagrams

In the OSI model, the UDP protocol exploits the IP services. The UDP service is connectionless, it doesn't get any logical connection between the two communicating hosts, it doesn't guarantee the sequence of the Informative Unit transfer on the net, finally there's no certainty about the correct datagrams transfer. One of the "vulnerabilities" of the UDP is the possibility to send "spoofed" packets (with a faked source IP).

This is the direct consequence of the connectionless service, realized by UDP.

3.1 Pieces of code

In order to make this explanation clear, here is a piece of code that let you spoof the source IP of the datagram.

```

.....
#include .... main()

{

int ....;

char ....;

```

```

int sd;

int port = 34567;

struct sockaddr_in source_addr, destination_addr;

udp_initialize(&source_addr, 0, inet_addr(argv[1])); //
(1)

udp_initialize(&destination_addr, port, inet_addr(argv[2])); //
(2)

sd = socket(AF_INET, SOCK_DGRAM, 0);

bind(sd, (struct sockaddr *) &source_addr,
      sizeof(source_addr));

sendto(sd, argv[3], strlen(argv[3]), 0, (struct sockaddr *)
       &destination_addr, sizeof(destination_addr));

void udp_initialize(struct sockaddr_in *address, int port, long
IPAddr)
{
address->sin_family = AF_INET;
address->sin_port = htons((u_short)port);
address->sin_addr.s_addr = IPAddr;
}

```

3.2 Explaining the code

(1) This piece of code calls the `udp_initialize` procedure that let you fill the socket structure fields. In this particular case we put the socket's port to 0 and the IP address as `argv[1]`. W.N: The Ip and the port are chosen arbitrarily, because no reply is expected.

In the case (2), the destination's socket value are set correctly.

The step (1) let us create a spoofed datagram.

Now that we have an idea of UDP's functions, we can understand my tool better. This program can be used in different ways, but in the complex it's surely interesting.

4 Direct contact with my implementation

The program was idealized in 2002 as a tool to execute remotely commands, without opening any direct logical connection with the wished server. I discarded immediately the TCP protocol, because it is connection-oriented. Another aspect that I wouldn't exclude was the possibility to avoid to open TCP ports, that could be easily scanned and maybe flooded (or similar). The first version of the program included a not very good dynamic algorithm, because it didn't supply any procedure of commands reading from an external file, but directly from the source file. For these reasons I coded another procedure that let you read commands to be executed from the configuration file.

4.1 Analysis of the Program

(This is my own program but you can create another one, this is only a concept!)

```
$cat server.c
```

```
#define PORT 32980          // Listening port.
```



```
#define ETC 'udp.conf'    // Configuration file. It indicates the
```



```
name
```



```
of the file, from which the reading of the
```



```
commands to execute will start.
```

You can modify the above values, as you wish.

The file "udp.conf" (not crypted), must be edited, for a correct use as follows:

```
$cat udp.conf
```

KEY WORD:/PATH/TO/PROGRAM

example:
angelo:/home/angelo/hello

When the server will receive the key-word "angelo", it will execute the file "/home/angelo/hello"

The tool even implements a logging procedure of all the received commands, with date and source IP.

```
fd = open("server.log", O_CREAT | O_RDWR | O_APPEND, 0644);
```

The log file will be placed in the same directory of the program and it's name will be "server.log".

During the implementation of the project, I have decided to crypt the configuration file and the log file too. The crypt() algorithm is really easy; it executes a XOR cryptation of any file's byte with the key.

The default key is:

```
int key = 0xff17261; // This is the cryptation Key; Change IT!!!
```

You can modify it and insert another one like 0xff71678 and so on...

The "UDP.CONF" file, after being edited correctly, must be crypted with the crypto program, that you will find in the main directory, of the package. It's really IMPORTANT that the keys of server.c and crypto.c must be the same, or the server program won't work at all, as it couldn't read the "udp.conf" file correctly.

4.2 Fast how to

Edit the "udp.conf" file following the above indications. Change the keys of server.c and crypto.c (they must be IDENTICAL!).

Compile the program.

\$/server (the program will go in background)

Now just send key commands using the client.

\$/client SOURCE-IP DEST-IP PORT command

5 Source of the project

Project Sources for Linux.

5.1 Server sources

```
##### SERVER.C #####
```

```
/*
```

```
***
```

```
***
```

```
***                UDP REMOTE CONTROLS
```

```
***
```

```
***
```

```
***                Copyright (c) Rosiello Security
```

```
***                All Rights Reserved
```

```
***                AUTHOR: Angelo Rosiello
```

```
***
```

```
***
```

```
***
```

```
***
```

```
*/
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <assert.h>
#include <netdb.h>
#include <time.h>
#include <stddef.h>

#define PORT 32980
#define ETC 'udp.conf'

void bg();
void udp_initialize();
char crypto(char ch);
void logging();

main() /* well, I like to put main() right here ;P */
{

    int sd, client_len, status, binding, fd, result, i, comando,
eof;

    char ch[1024], temp, crypted, memorizza[100];
    struct sockaddr_in server_addr, client_addr;
```

```

udp_initialize(&server_addr, PORT, INADDR_ANY);

sd = socket(AF_INET, SOCK_DGRAM, 0);

if(sd < 0) perror('Socket');

assert(sd >= 0);

binding = bind(sd, (struct sockaddr *) &server_addr,
               sizeof(server_addr));

if(binding != 0) perror('Bind');

assert(binding == 0);

fd = open(ETC, O_RDONLY);

if(fd <= 0) perror('UDP.CONF');

assert(fd > 0);

bg();

while(1)
{
    for(i = 0; i < 1024; i++) ch[i] = '';
    for(i = 0; i < 100; i++) memorizza[i] = '';
    recvfrom(sd, ch, 1023, 0, (struct sockaddr *)
&client_addr,
            &client_len);

    result = 1;

    i = 0;

    lseek(fd, 0, 0);

    logging(ch);

    while(result)
    {
        eof = read(fd, &temp, 1);

```

```
if(eof == 0) break;
crypted = crypto(temp);
if(crypted != ':' && crypted != 'n')
{
    memorizza[i] = crypted;
    i++;
}
else if(crypted == 'n')
{
    for(i=0; i< 100; i++) memorizza[i] =
'';

    i = 0;
}
else if(crypted == ':')
{
    if(!strcmp(memorizza, ch))
    {
        for(i=0; i< 100; i++)
            memorizza[i] =
'';

        i = 0;
        comando = 1;
        while(comando)
        {
            read(fd, &temp, 1);
            crypted = crypto(temp);

            if(crypted != 'n')
            {
                memorizza[i] = crypted;
```

```
        i++;
    }
    else if(cripted ==
'n')
    {
system(memorizza);
        comando = 0;
    }
}
else
{
    for(i=0; i<100; i++)
memorizza[i]= '';
        i = 0;
    }
}
}
close(fd);
}
/*
```

```
** Implementing the procedures
**
** 1) Background
**
** 2) Setting up addresses
**
** 3) Logging
**
** 4) Decrypting the config file and reading it
**
*/
```

```
/******
* Background procedure *
* Nothing to explain even... *
*****/
```

```
void bg()
{
    signal(SIGHUP, SIG_IGN);
    if (fork () != 0)
    {
        exit(0);
    }
}
```

```
/******  
* Address initialize procedure *  
*****/  
  
void udp_initialize(struct sockaddr_in *address, int port, long IPaddr)  
{  
    address->sin_family = AF_INET;  
    address->sin_port = htons((u_short)port);  
    address->sin_addr.s_addr = IPaddr;  
}  
  
/******  
* Crypting procedure *  
* The configuration file will be decripted *  
* The Log file will be cripted... *  
*****/  
  
char crypto(char ch)  
{  
    char crypted;  
    int key = 0xff17261;  
    crypted = ch ^ key;  
    return(crypted);  
}
```

```

/*****
* Logging procedure                                     *
* The Logs' file will be 'server.log'                 *
*****/

void logging(char ch[1024])
{
    struct hostent *entity;
    struct sockaddr_in client_addr;
    struct tm *ptr;
    int fd, i;
    unsigned long ip;
    char orario[30], temp[1024], crypted, carattere, indirizzo[32];
    char source[] = 'Connection from n';
    char stringa[] = ' ---> Received: n';
    time_t lt;
    lt = time(NULL);
    ptr = localtime(<);
    fd = open('server.log', O_CREAT | O_RDWR | O_APPEND, 0644);
    if(fd < 0) perror('Opening logs' file');
    assert(fd >= 0);
    entity = gethostbyaddr((char *) &client_addr.sin_addr,
        sizeof(struct in_addr), AF_INET);
    assert(entity >= 0);
    ip = inet_ntoa((struct in_addr) client_addr.sin_addr);
    i = 0;

    sprintf(orario, '%s', (asctime(ptr)));

```

```
while(1)
{
    if(orario[i] == 'n')
    {
        i = 0;
        break;
    }
    carattere = orario[i];
    crypted = crypto(carattere);
    write(fd, &crypted, 1);
    i++;
}
crypted = crypto('n');
write(fd, &crypted, 1);

while(1)
{
    if(source[i] == 'n')
    {
        i = 0;
        break;
    }
    carattere = source[i];
    crypted = crypto(carattere);
    write(fd, &crypted, 1);
    i++;
}

sprintf(indirizzo, '%sn', ip);
while(1)
```

```
{  
  
    if(indirizzo[i] == 'n')  
    {  
  
        i = 0;  
  
        break;  
  
    }  
  
    carattere = indirizzo[i];  
    crypted = crypto(carattere);  
    write(fd, &crypted, 1);  
    i++;  
}  
  
while(1)  
{  
  
    if(stringa[i] == 'n')  
    {  
  
        i = 0;  
  
        break;  
  
    }  
  
    carattere = stringa[i];  
    crypted = crypto(carattere);  
    write(fd, &crypted, 1);  
    i++;  
}  
  
while(1)  
{  
  
    if(ch[i] == '')  
    {  
  
        i = 0;
```

```

        break;

    }

    carattere = ch[i];
    crypted = crypto(carattere);
    write(fd, &crypted, 1);
    i++;
}

crypted = crypto('n');
write(fd, &crypted, 1);
write(fd, &crypted, 1);
close( fd );
}

```

```

/*
**
**                               EOF
**
**
**
*/

```

5.2 Client Sources

```

##### CLIENT.C
#####

/*
**
**
**                               Unix Udp Client
**
**
**      This is the Unix client UDP , to send spoofed/unspoofed
commands.

```

```
**  
** ex:  
** $. /udp source-ip destination-ip dest-port message  
**  
**  
** AUTHOR: Angelo Rosiello  
**  
*/  
  
#include <stdio.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <string.h>  
#include <assert.h>  
  
#define mia_porta 0  
  
void udp_initialize();  
  
int main(int argc, char *argv[])  
{  
    int sd, PORT, binding;  
    struct sockaddr_in server_addr, mio_addr;  
    if(argc != 5)  
    {  
        fprintf(stdout, 'Unix UDP client by Angelo Rosiellon');  
        fprintf(stdout, 'USAGE: %s SOURCE-IP DEST-IP PORT MESSAGEn',  
                argv[0]);  
        exit(0);  
    }  
}
```

```
    }

    PORT = atoi(argv[3]);

    udp_initialize(&mio_addr, mia_porta, (long)
inet_addr(argv[1]));

    udp_initialize(&server_addr, PORT, (long) inet_addr(argv[2]));

    sd = socket(AF_INET, SOCK_DGRAM, 0);

    binding = bind(sd, (struct sockaddr *)
                    &mio_addr, sizeof(mio_addr));

    if(binding != 0) perror('Bind');

    assert(binding == 0);

    sendto(sd, argv[4], strlen(argv[4]), 0, (struct sockaddr *)
            &server_addr, sizeof(server_addr));
}

void udp_initialize(struct sockaddr_in *address, int port, long IPAddr)
{
    address->sin_family = AF_INET;

    address->sin_port = htons((u_short)port);

    address->sin_addr.s_addr = IPAddr;
}

/*
***
***   EOF
***
*/
```

5.3 Crypto sources

```
##### CRYPTO.C
#####

/*
***          Crypto Program
***
***          Copyright (c) Rosiello Security
***          All Rights Reserved
***          AUTHOR: Angelo Rosiello
***
*/

#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <assert.h>
#include <unistd.h>
#include <sys/types.h>

char crypto(char ch);

main(int argc, char *argv[])
{
    char ch, crypted;
    int fd, new_fd, lettura;
    if(argc != 3)
    {
        printf('Copyright (c) 2002/03 Angelo Rosiellon');
        printf('All Rights Reservedn');
    }
}
```

```

        printf('USAGE: %s SOURCE-FILE DEST-FILEn');
        exit(0);
    }

    fd = open(argv[1], O_RDONLY);
    assert(fd > 0);
    if(fd <= 0) perror('file');
    new_fd = open(argv[2], O_CREAT | O_RDWR | O_TRUNC, 0644);
    assert(new_fd > 0);
    if(new_fd <= 0) perror('file');
    while(1)
    {
        lettura = read(fd, &ch, 1);
        if(lettura == 0) break;
        crypted = crypto(ch);
        write(new_fd, &crypted, 1);
    }
    close( fd );
    close( new_fd );
}

char crypto(char ch)
{
    char crypted;
    int key = 0xff17261;
    crypted = ch ^ key;
    return(crypted);
}

/*

```

*** EOF

*/

END OF PROJECT

Reference

Rosiello Security - <http://www.rosiello.org>