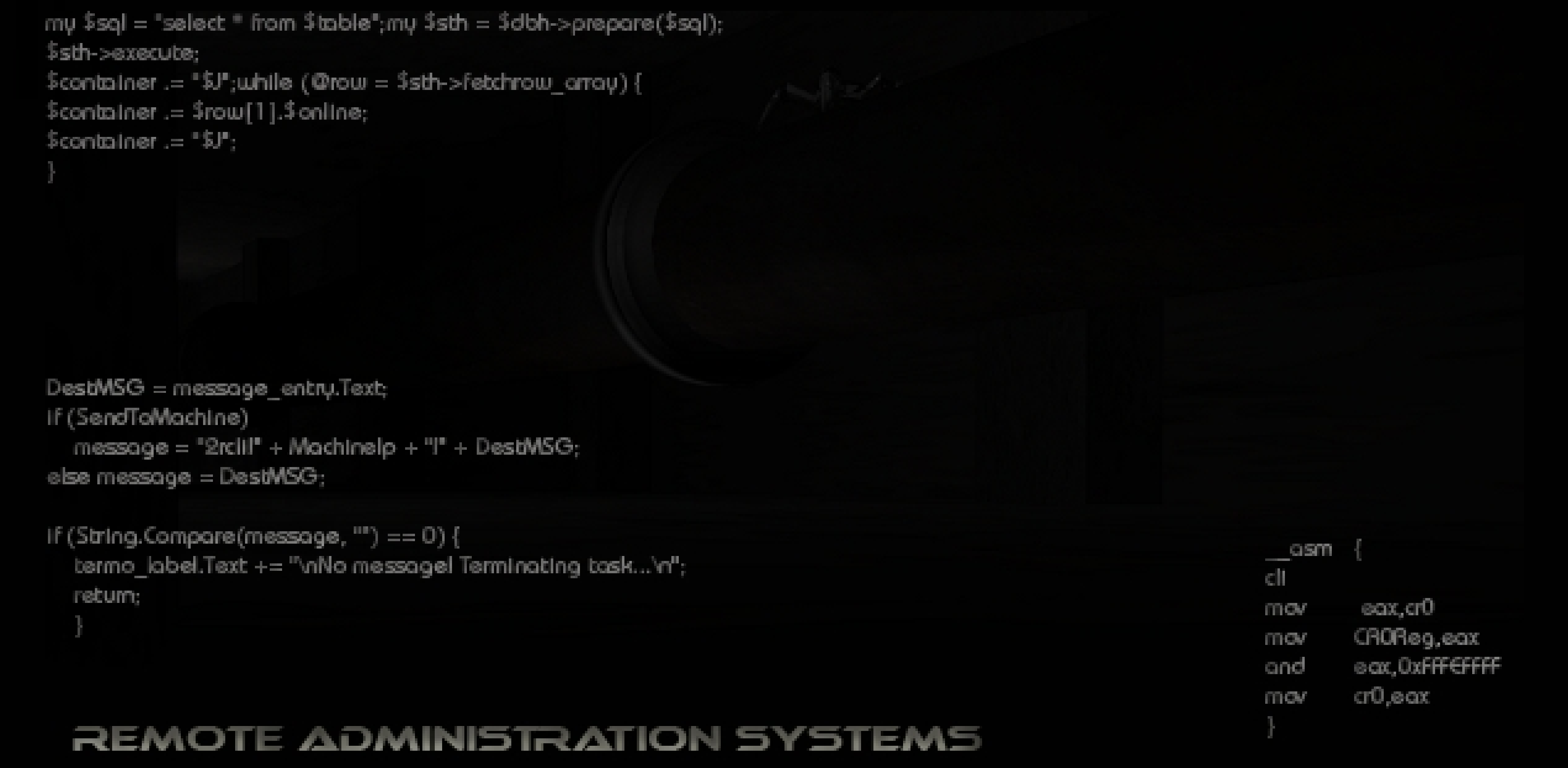


```
my $sql = "select * from $table";my $sth = $dbh->prepare($sql);
$sth->execute;
$container .= "$J";while (@row = $sth->fetchrow_array) {
$container .= $row[1].$online;
$container .= "$J";
}
```

```
DestMSG = message_entry.Text;
if (SendToMachine)
   message = "2rcl|" + MachineIp + "|" + DestMSG;
else message = DestMSG;

if (String.Compare(message, "") == 0) {
   termo_label.Text += "\nNo message! Terminating task...\n";
   return;
   }
```

```
__asm   {
cli
mov       eax,cr0
mov      CR0Reg,eax
and      eax,0xFFFEFFFF
mov      cr0,eax
}
```

REMOTE ADMINISTRATION SYSTEMS

xımachine project proudly presents :

"Introduction to Remote Administration Systems"

by cross

#define ARTICLE_STATUS "0.03%"

Remote Administration refers to any method of controlling a computer from a remote location.

Software that allows remote administration is becoming increasingly common and is often used when it is difficult or impractical to be physically near a system in order to use it, or in order to access web material that is not available in one's location, for example viewing the BBC iPlayer from outside the United Kingdom. A remote location may refer to a computer in the next room or one on the other side of the world. It may also refer to both legal and illegal remote administration.

*** Wikipedia

Here we are again and its time to throw some light on something more advanced then a remote shell. In this article i will present one of many implementations of its structure and small example of its functionality. So lets skip unnecessary stories and get one step closer to the subject.

*** cross

| Prerequisites |

To get a point about whole thing that will be discussed here you need some basic knowledge about:

1. Linux and Windows systems
2. c/c++ programming for windows and linux
3. Perl scripting language
4. Windows kernel mode drivers development
5. network programming
6. programming kernel mode network clients / Winsock Kernel
7. Gtk+ GUI programming
8. C# .NET programming

Tools. However these tools i have used and i am not forcing you to follow me step by step – our main task could be accomplished in a different ways:

1. Windows Driver Kit version >= 6
2. Phisical Linux with gcc compiler and Gtk+ gui (development + runtime) libs sat up nicely
3. Phisical Windows with Visual Studio Pro 2008 and Perl interpreter
4. VMWare workstation / server
5. Virtual Linux System with perl interpreter
6. Virtual Windows System (Vista / Server 2008 / 7)
7. some Driver Loader

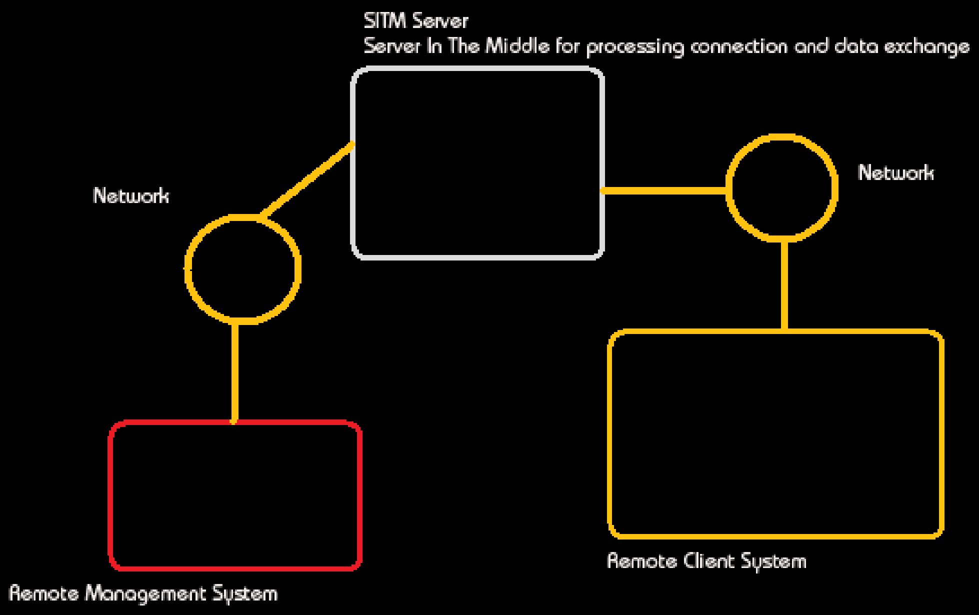
Final 'like in real life' test will require this. However, you can pack and mix all of this into one Windows System (Vista / Server 2008 / 7). A few words before we start. This project – is not something big and scary. I have got in plans to make something 4x times bigger but resigned due to many facts. Lack of time for example. These days we all live fast, want to do a lot, want to learn, want to get more knowledge – life is not enough. Here i will show a 'skeleton', and if you will find it somehow interesting, usefull, if this will bring you some new ideas – then you have a good base to start from, make it bigger. Another thing, Remote Administration Systems are often associated with malicious software, trojans for example, botnets, etcetera. How you will use my code – it is up to you. You can make smth legit from it and there are many examples of legit software which uses same functionality. And many examples of botnets.

While the term "botnet" can be used to refer to any group of bots, such as IRC Bots, this word is generally used to refer to a collection of compromised computers (called Zombie computers) running software, usually installed via drive-by downloads exploiting Web browser vulnerabilities, worms, Trojan horses, or backdoors, under a common command-and-control infrastructure.

*** Wikipedia

Lets proceed with caution ;)

| Structure |



Ehm, the whole thing will have 'triangle like' structure. Consider about following components:

1. Master machine - machine from which we will establish control
2. Server machine - here we will place our server (this point will be explained later)
3. Client machine - machine which will be under our control / administration

Master Machine <---> Server Machine <---> Client Machine

Now why such structure? Lets assume that i am remotely administrating in real time 100 machines, anyways, number could be bigger. I want to know (or i need to know) what is happing on each machine. The easiest way to make it real - force every machine to send status information to my personal computer at home. But how about the fact, that maybe i dont have to much bandwidth or i got limited amount of traffic, besides, not every message is critical and require my attention - some of messages i can simply check out a little bit later and i want to receive only critical reports in real time?
To avoid creating a situation, in which a lot of 'trash' traffic will come to my personal network, i will implement 'Server In The Middle', a server between me and client machine, which will decide what to send to me.

I hope you got the point. Now, how i will code all this.
First of, if we are going to administrate / control something , we need some nice user interface for it. Mate, i am not going to mess with console application, no way. Here we need our handy skills in creating GUI interfaces. Second, which OS we got? Linux? Windows? Well, i got both. Assuming that i am working on Linux - i will use Gtk+ as my GUI library for my Administration Console; lets name it now, R-manager for example. Do you like its new name? ;) You can use some other if wish. Ok, thats clear for now. But how about sysadmins, which work on windows boxes? Lets not forget about them and code another version of R-manager, this time we will use C# .NET. Or you can compile Gtk+ version on windows - you just need libs.
So we have to code 2 versions of R-manager, clear.
Work name: R-manager.

Now what about 'Server In The Middle'? By the way - this is its name, my friends, - SITM Server.

I have chosen Perl scripting language for this task. Earlier i said that you can use just one op. system for testing this project and i havent lied. You can run this Server on windows box as well - just need to setup perl interpreter on windows machine and thats all! Oh, one thing will be unavailable - server will not run as daemon (in windows terms - service) unless you will be able to setup Daemon perl module. Ok.

Work name: SITM Server.

Now, client machine. Our client machine is running windows server 2008. Now we have to code an application which will provide some control over this machine for us. I could have it done with usage of Win32 API, code simple ring3 app and so what? Where is some fun in it? Boring, i am bored to death...

That is why my client app will be kernel mode network driver. Actually that is even better! While we are in kernel, we got a lot more fun and, we got total control over client machine. And that is good apportunity to have some practice with Winsock Kernel, WSK, - new NPI by MS.

Work name: R-client

Alright, we know what we have to do. We know it is easy task. We are doing it for fun and practice.

This examples here will represent minimum of functionality, basic things, a skeleton lets say.

Lets proceed to a next chapter.

__| R-client |__

So, what functionality we are going to implement? Lets do something simple like this:

After R-client (driver) is initialized it will wait for our 'order' to protect itself from unload by setting up hook on single function - NtUnloadDriver. If user will try to uload our driver with usage of the above function - R-client will send a message to us, informing about unload attempt. Simple isn't it?

Then we will code such small app and test it in real life.

So our driver will receive messages from us and send messages to us. I mean, to SITM Server.

In some cases it will, be encrypting outcomming messages with simple rot47 cipher and decrypting incomming.

It will use UDP protocol for this and will be acting as client and server.

And last, it will hook NtUnloadDriver. Now we got full picture of our R-client driver.

1. Network based messaging system
2. Encryption / decryption
3. Kernel mode hook

Lets start from the very last point.

NtUnloadDriver kernel mode hook.

his we will accomplish by inserting an unconditional 5-byte jump into original NtUnloadDriver function entry point what will result in redirection execution of NtUnloadDriver to our function, lets say:
NewNtUnloadDriver. And then our NewNtUnloadDriver will decide what to do next. I am not getting into details, but lets just clarify a few things.

Our function is pretty trivial:

```
NTSTATUS __stdcall NewZwUnloadDriver(IN PUNICODE_STRING DriverServiceName){
WCHAR Path[1024];
RtlZeroMemory(Path, sizeof(Path)); // here we will store reg path of driver service
swprintf(Path, L”%wZ”, DriverServiceName); // store it actually
if(wcscmp(Path, L”\\Registry\\Machine\\SYSTEM\\CurrentControlSet\\Services\\test.sys”) == 0){
//^ if registry path  = our driver path -> access denied and send message to server
DbgPrint(”R-client: Access denied!\n”);
SendMessage(”rcli!Something tried to unload me”);
return STATUS_ACCESS_DENIED;
} else { // just execute original NtUnloadDriver
DbgPrint(”DriverServiceName: %wZ”, DriverServiceName);
return OriginalZwUnloadDriver(DriverServiceName);
        }
}
```

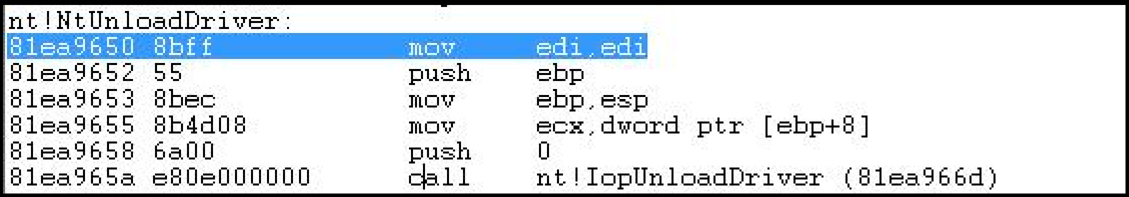
By reading comments in code you see how it works. Further explanation i will split into following points:

1. Identify original function structure
2. Machine code before
3. Hook implementation
4. Machine code after
5. Conclusion

1. NtUnloadDriver C code:

```
NTSTATUS NTAPI NtUnloadDriver (IN PUNICODE_STRING DriverServiceName){

return IopUnloadDriver(DriverServiceName, FALSE);

}
```

2. Machine code before



```
nt!NtUnloadDriver:
81ea9650 8bff            mov     edi,edi
81ea9652 55              push    ebp
81ea9653 8bec            mov     ebp,esp
81ea9655 8b4d08          mov     ecx,dword ptr [ebp+8]
81ea9658 6a00            push    0
81ea965a e80e000000      call    nt!IopUnloadDriver (81ea966d)
```

3. ==> Locate space for 'jmp' insertion
   ==> Calculate Callgate
   ==> Memory allocation
   ==> Save original function address
   ==> Insert 'jmp' to our new function
      => Disable kernel memory protection
      => Insert 'jmp'
      => Enable memory protection

To accomplish this task we use *Length-Disassembler Engine* by (C)ZOMbiE (included within the project).

// locating place to set our 5-byte jump

```
while (CollectedSpace < 5){
        GetInstLenght(Inst, &Size); // thanks ZOMbiE
        (unsigned long)Inst += Size; // next instruction
        CollectedSpace += Size; // update space
}
```

// memory allocation

```
CallGateSize = CollectedSpace + 5;
CallGate = (void *)ExAllocatePool(NonPagedPool, CallGateSize);
```

// clear memory with nope's (0x90)

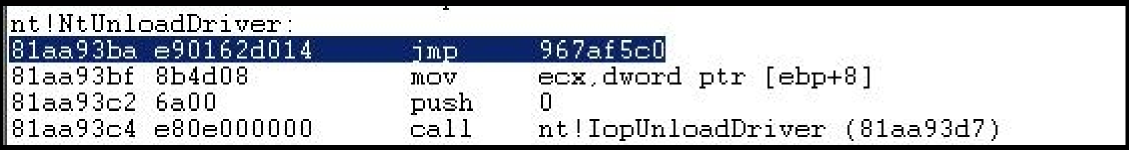
```
memset(CallGate, 0x90, CallGateSize);
memcpy(CallGate, Addr, CollectedSpace);
memset(Addr, 0x90, CollectedSpace);
```

// generate jump

```
*((unsigned long *)(((unsigned long)CallGate + CollectedSpace) + 0x1)) =
(((unsigned long)Addr + 0x5) -        ((unsigned long)CallGate + CollectedSpace) - 0x5);
*((unsigned char *)((unsigned long)CallGate + CollectedSpace)) = 0xe9;
*((unsigned long *)(((unsigned long)Addr) + 0x1)) = (((unsigned long)NewFunc) -
((unsigned long)Addr)          - 0x5);
*((unsigned char *)((unsigned long)Addr)) = 0xe9;
```

```
// typedefs

typedef NTSTATUS (* _ZwUnloadDriver)(IN PUNICODE_STRING DriverServiceName);
_ZwUnloadDriver OriginalZwUnloadDriver;

// disable kernel memory protection

__asm
{
        cli
        mov             eax,cr0
        mov             CR0Reg,eax
        and             eax,0xFFFEFFFF
        mov             cr0,eax
}

// Hook

OriginalZwUnloadDriver =
(_ZwUnloadDriver)SetHook(
(PVOID)KeServiceDescriptorTable.ServiceTableBase[*(PULONG)((ULONG)(ZwUnloadDriver)+1)],
NewZwUnloadDriver
);

// Enable kernel memory protection

__asm
{
        mov             eax,CR0Reg
        mov             cr0,eax
        sti
}
```

4. Machine code after



```
nt!NtUnloadDriver:
81aa93ba e90162d014       jmp     967af5c0
81aa93bf 8b4d08           mov     ecx,dword ptr [ebp+8]
81aa93c2 6a00             push    0
81aa93c4 e80e000000       call    nt!IopUnloadDriver (81aa93d7)
```

5. Conclusion

After all we have original function address and can jump to it whenever we wish from our NewUnloadDriver. With this template and thanks to ZOMbiE's disasm engine, you can implement your own kernel mode hooks. You can hide files, folders, forbid access to any place you want, manipulate remote system the way you want. I have chosen to implement such technique in this project as it was the most suitable to show how we can create simple monitoring routines.

X. Testing.

For testing purposes of the above method, i am providing Win32 application.
Source (Unload.c):

```
#include <windows.h>
#define SystemLoadAndCallImage 38
#define  STATUS_SUCCESS                    ((NTSTATUS)0x00000000L)
//#define  STATUS_SUCCESS                    ((NTSTATUS)0x00000000L)
typedef  LONG KPRIORITY;
#define  NtGetProcessHeap()                 (Nt_CurrentTeb()->PebBaseAddress->DefaultHeap)
#define  STATUS_INFO_LENGTH_MISMATCH       ((NTSTATUS)0xC0000004L)
#define  NtCurrentThread()                  ((HANDLE) -2)

//======================== NT definitions ============================
typedef struct _STRING {
    USHORT Length;
    USHORT MaximumLength;
#ifdef MIDL_PASS
    [size_is(MaximumLength), length_is(Length) ]
#endif // MIDL_PASS
    PCHAR Buffer;
} STRING, *PSTRING;
typedef STRING ANSI_STRING;
typedef PSTRING PANSI_STRING;
typedef STRING OEM_STRING;
typedef PSTRING POEM_STRING;
typedef struct _CLIENT_ID{
    HANDLE UniqueProcess;
    HANDLE UniqueThread;
} CLIENT_ID, *PCLIENT_ID;

#if (_WIN32_WINNT >= 0x0400)
#define EXIT_STACK_SIZE 0x188
#else
#define EXIT_STACK_SIZE 0x190
#endif

typedef struct _UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWSTR  Buffer;
} UNICODE_STRING;
typedef UNICODE_STRING *PUNICODE_STRING;
typedef long NTSTATUS;
#define NT_SUCCESS(Status) ((NTSTATUS)(Status) >= 0)

typedef enum {
     AdjustCurrentProcess,
     AdjustCurrentThread
} ADJUST_PRIVILEGE_TYPE;


NTSTATUS (NTAPI *NtUnloadDriver)( IN PUNICODE_STRING DriverServiceName );
VOID (NTAPI *RtlInitUnicodeString)(PUNICODE_STRING DestinationString,PCWSTR SourceString);
LONG (__stdcall *RtlAdjustPrivilege)(int,BOOL,BOOL,BOOL *);

BOOL NtFunctionsInit(){
     const char NTDLL[] = { 0x6e, 0x74, 0x64, 0x6c, 0x6c, 0x2e, 0x64, 0x6c, 0x6c, 0x00 };
     HMODULE hObsolete                    = GetModuleHandle(NTDLL);
     *(FARPROC *)&RtlInitUnicodeString      = GetProcAddress(hObsolete, "RtlInitUnicodeString");
     *(FARPROC *)&NtUnloadDriver                 = GetProcAddress(hObsolete, "NtUnloadDriver");
     *(FARPROC *)&RtlAdjustPrivilege          = GetProcAddress(hObsolete, "RtlAdjustPrivilege");
return 0;
}

int main(){
    BOOL en;
    UNICODE_STRING u_str;
    WCHAR RegUniPath[MAX_PATH] =
L"\\Registry\\Machine\\SYSTEM\\CurrentControlSet\\Services\\test.sys";
    NtFunctionsInit();
    RtlAdjustPrivilege(10, TRUE, AdjustCurrentProcess, &en);
    RtlInitUnicodeString(&u_str, RegUniPath);
    if(NtUnloadDriver(&u_str) != STATUS_SUCCESS)MessageBox(0, "Unable to unload driver!",
"ERROR",     MB_ICONERROR);
 return 0;
}
```

Could be compiled with Dev-Cpp or MS Visual Studio.

_| Encryption / Decryption.

Well, that is almost unnecessary thing in our project due to a lot of things that still remains not shown, however:

```
char *code_rot(char *cod_dec){
    char *p = cod_dec;

while(*p) {
        if(*p >= ’!’ && *p <= ’O’)
        *p = ((*p + 47) % 127);
        else if(*p >= ’P’ && *p <= ’~’)
        *p = ((*p - 47) % 127);
        p++;
    }
return cod_dec;
}
```

As simple as it gets, my friends :)

_| Network based messaging system.

Winsock Kernel (WSK) is a kernel-mode Network Programming Interface (NPI).With WSK, kernel-mode software modules can perform network I/O operations using the same socket programming concepts that are supported by user-mode Winsock2. The WSK NPI supports familiar socket operations such as socket creation, binding, connection establishment, and data transfers (send and receive). However, while the WSK NPI supports most of the same socket programming concepts as user-mode Winsock2, it is a completely new and different interface with unique characteristics such as asynchronous I/O that uses IRPs and event callbacks to enhance performance.

*** MSDN

As part of introduction, let me present a small framework, provided by (C)MaD, which consist of following
function wrappers, making kernel sockets programming easier then ever.

```
NTSTATUS NTAPI SocketsInit();
VOID NTAPI SocketsDeinit();

PWSK_SOCKET NTAPI
CreateSocket(
    __in ADDRESS_FAMILY         AddressFamily,
    __in USHORT                 SocketType,
    __in ULONG                  Protocol,
    __in ULONG                  Flags
    );

NTSTATUS NTAPI
```

```
CloseSocket(
        __in PWSK_SOCKET WskSocket
        );

NTSTATUS NTAPI
Connect(
        __in PWSK_SOCKET        WskSocket,
        __in PSOCKADDR          RemoteAddress
        );

PWSK_SOCKET NTAPI
SocketConnect(
        __in USHORT                 SocketType,
        __in ULONG          Protocol,
        __in PSOCKADDR  RemoteAddress,
        __in PSOCKADDR  LocalAddress
        );

LONG NTAPI
Send(
        __in PWSK_SOCKET        WskSocket,
        __in PVOID                  Buffer,
        __in ULONG                  BufferSize,
        __in ULONG                  Flags
        );

LONG NTAPI
SendTo(
        __in PWSK_SOCKET        WskSocket,
        __in PVOID                  Buffer,
        __in ULONG                  BufferSize,
        __in_opt PSOCKADDR      RemoteAddress
        );

LONG NTAPI
Receive(
        __in  PWSK_SOCKET       WskSocket,
        __out PVOID                 Buffer,
        __in  ULONG                         BufferSize,
        __in  ULONG                         Flags
        );

LONG NTAPI
ReceiveFrom(
        __in  PWSK_SOCKET       WskSocket,
        __out PVOID             Buffer,
        __in  ULONG                     BufferSize,
        __out_opt PSOCKADDR     RemoteAddress,
        __out_opt PULONG ControlFlags
        );

NTSTATUS NTAPI
Bind(
        __in PWSK_SOCKET        WskSocket,
        __in PSOCKADDR          LocalAddress
        );

PWSK_SOCKET NTAPI
Accept(
        __in PWSK_SOCKET        WskSocket,
        __out_opt PSOCKADDR     LocalAddress,
        __out_opt PSOCKADDR     RemoteAddress
  );
```

We are not going to explore the most deepest aspects of Winsock Kernel here, because it is not the point. To implement any type of connection-oriented socket using WSK in our example, we first need:

- Register winsock kernel application
- Create socket
- Bind socket to a local transport address

R-Client use UDP protocol for network communication with SITM Server, then our socket will be a datagram
socket. After binding to a local address, driver creates a system thread, which then receives in a loop datagrams and depending on incomming messages from Server – performs some action. For example:

1. Got message "hi" ---> send message "Hello :)"

Lets see some code at last ;)

```
// Initialization of networking

NTSTATUS InitNetworking(){
NTSTATUS    Status = STATUS_UNSUCCESSFUL;
SOCKADDR_IN LocalAddress;

Status = SocketsInit(); // socket init
if (!NT_SUCCESS(Status)) {            // if failed
        DbgPrint("SocketsInit() failed with status 0x%08X\n", Status);
        PsTerminateSystemThread(Status);
} else DbgPrint("Kernel Sockets Initialized successfuly!\n");

g_ServerSocket = CreateSocket(AF_INET, SOCK_DGRAM, IPPROTO_UDP,
WSK_FLAG_DATAGRAM_SOCKET); // socket creation
if (g_ServerSocket == NULL) {
        DbgPrint("CreateSocket() returned NULL\n");
        PsTerminateSystemThread(Status);
} else DbgPrint ("Socket created!\n");

// just like Winsock2 API
LocalAddress.sin_family                 = AF_INET;
LocalAddress.sin_addr.s_addr        = inet_addr(RCLI_ADDR);
LocalAddress.sin_port                   = HTONS(SERVER_PORT);

Status = Bind(g_ServerSocket, (PSOCKADDR)&LocalAddress); // binding to local address
if (!NT_SUCCESS(Status)) { // if failed ...
        DbgPrint("Bind() failed with status 0x%08X\n", Status);
        CloseSocket(g_ServerSocket);
        g_ServerSocket = NULL;
        PsTerminateSystemThread(Status);
} else DbgPrint("Binded to local address...\n");

return STATUS_SUCCESS;
}
```

NOTE* g_ServerSocket is our global variable and now is our fresh and shining, initialized datagram socket for later use. And thats it! Now R-Client driver can receive and send messages with usage of "g_ServerSocket" until it is closed.
Finally, some fresh air needed. Lets make our way into the network...

```
NTSTATUS SendMessage(char *message){
NTSTATUS    Status = STATUS_UNSUCCESSFUL;
SOCKADDR_IN ServerAddr, LocalAddress;
char encoded[1024];
RtlZeroMemory(encoded, sizeof(encoded));
strcpy(encoded, message);
code_rot(encoded);     // encrypting message with rot47 cipher, just for example
ServerAddr.sin_family                    = AF_INET;
ServerAddr.sin_addr.s_addr  = inet_addr(SERVER_ADDR);        // address
ServerAddr.sin_port              = HTONS(6666); // SITM Server port

Status = SendTo(g_ServerSocket, encoded, strlen(encoded), (PSOCKADDR)&ServerAddr);
if (!NT_SUCCESS(Status)) {
        DbgPrint("Could not send data! Status 0x%08X\n", Status);
        PsTerminateSystemThread(Status);
} else DbgPrint("Data sent!\n");

return STATUS_SUCCESS;
}


static VOID UdpServer(PVOID Context){
.................
SendMessage(status); // datagram, sent here, containes some initial information
// based on which SITM decides if our machine is already registered in a system or not,
// it containes binded R-Client's port, etc, etc, whatever information we would like to send
// during startup
......................................
.....................................
while(1){
Status = ReceiveFrom(g_ServerSocket, response, 1024, NULL, NULL);
// a loop
if(strcmp(response, "hook") == 0){ // if we got order to setup hook
DbgPrint("Setting up hooks...\n");
KeDelayExecutionThread(KernelMode,FALSE,&Interval ); // take a breath
   SetHooks(); // set it
        SendMessage("rcli!NtUnloadDriver Hooked"); // send message
        } else if(strcmp(response, "hi") == 0){          // just say hi to master :)
        KeDelayExecutionThread(KernelMode,FALSE,&Interval );
        SendMessage("rcli!Hello :)"); // send
        }
}
```

You may ask about "rcli!" at the beginning of message. Well, it informs server that message came from R-Client, exclamation mark is splitter, and the rest, what is after is actual message.

_Begin: (Receive ==> Process ==> Send) goto _Begin;

Phew, done. R-Client explanation reached its final step.

```
#undef ARTICLE_STATUS
        #define ARTICLE_STATUS "43%"
```

Perl is a high-level, general-purpose, interpreted, dynamic programming language. Perl was originally developed by Larry Wall, a linguist working as a systems administrator for NASA, in 1987, as a general-purpose Unix Scripting language to make report processing easier. Since then, it has undergone many changes and revisions and become widely popular amongst programmers. Larry Wall continues to oversee development of the core language, and its upcoming version, Perl 6.

Perl borrows features from other programming languages including C, shell scripting, AWK, and sed. The language provides powerful text processing facilities without the arbitrary data length limits of many contemporary Unix tools, facilitating easy manipulation of text files. It is also used for graphics programming, system administration, network programming, applications that require database access and CGI programming on the Web. Perl is nicknamed "the Swiss Army chainsaw of programming languages" due to its flexibility and adaptability.

*** Wikipedia

I think that description speaks for itself and explains decision of choosing Perl as programming language for upcomming task. Need to code right away some fast and stable script – perl is your friend.
Perl was my first programming language, it stay with me until today and will stay until i die.

print " Hi there :) \n"; # we rock!

Now, focus. We are not going to waste a lot of time on this script 'coz we want it fast and we want it right away, right? Neither i will repeat myself about its mission.
The first of, after R-Client initial message appears, SITM Server checks in MYSQL Database – is there some records of just connected remote machine. In case of one machine, i mean, if we have just one remote machine under surveillance – i dont think there is a reason for playing with databases.
But if we are talking about 100 / 1000 machines, or even 10 – i think there is. Well, my question, have you created your database already?

```
my $dbhost = "localhost";
my $dbuser = "root";
my $dbpass = "";
my $table = "machines";
my $db = "project";

sub SetDB {
my $nodb = undef;
my $dbh_create = DBI->connect("dbi:mysql:$nodb:$dbhost",$dbuser,$dbpass) ; # connect to database
my $sql_create = "create database $db";        #query to execute
my $sth_create = $dbh_create->prepare($sql_create); # prepare query
$sth_create->execute or die "MYSQL error while creating Database! ($DBI::errstr)\n"; # execute it or die
# we are not going anywhere without our database
my $dbh = DBI->connect("dbi:mysql:$db:$dbhost",$dbuser,$dbpass) ;
my $sql = "create table `$table` (".
                                "`id` int(10) not null auto_increment,".
                                "`ip_as_name` varchar(150) not null default '',".
                                "`port` varchar(50) not null default '',".
                                " primary key(`id`),".
                                " unique key `id`(`id`)".
                                ") type=MyISAM comment='' AUTO_INCREMENT=1 ;";
my $sth = $dbh->prepare($sql); # prepare table creation query

$sth->execute or die "MYSQL error while creating table! ($DBI::errstr)\n"; # execute it or die
#not going anywhere neither if the above function failed
print "MYSQL Database setup finished\nDetailes:\nDatabase Host: $dbhost\nDatabase User: $dbuser\
Database Password: $dbpass\nTable: $table\n"; # print some stats
return "DBCREATED"; # return some value, in some case we may need it later
}
```

Thats my default settings, default fields and rows. We are still not even closer.

```
sub FetchMachines {
my ($ip_addr) = @_;  # that is IN paramater
my $online;
my $container; my $num; my $i;
$container .= "machines$J";
my $dbh = DBI->connect("dbi:mysql:$db:$dbhost",$dbuser,$dbpass) ;
my $sq = "select count(*) from $table";
my $st = $dbh->prepare($sq);
$st->execute or print("MYSQL error while fetching machines list! ($DBI::errstr)");
while (@row = $st->fetchrow_array){
$container .= $row[$i++];
}

my $sql = "select * from $table";
my $sth = $dbh->prepare($sql);
$sth->execute or print("MYSQL error while fetching machines list! ($DBI::errstr)");
$container .= "$J"; # $J is a splitter '!'
while (@row = $sth->fetchrow_array) {
$container .= $row[1].$online;
$container .= "$J";
}
&SendStatus ( "mcs", $J, "$container", $ip_addr, $SEND_PORT);
# $SEND_PORT – is a binded port of administrator's machine
}
```

Above routine performs trivial task:

1. Fetches machines from database
2. Fetches number of machines in database
3. NOTE *: In our example we are talking all the time about single one lonely machine ;)

And then it sends collected data to .. for example me.

```
sub AddMachine {
my ($ip_as_name, $port) = @_; # input parameters
my $trash = "Server~#";
my $add_err = "($DBI::errstr) while registering new machine!($ip_as_name)";
my $check_err = "MYSQL error ($DBI::errstr) while checking new machine ($ip_as_name)!";
print "registering ".$ip_as_name."\n";
my $dbh = DBI->connect("dbi:mysql:$db:$dbhost",$dbuser,$dbpass) ;
my $sql_ex = "select * from $table where ip_as_name = '$ip_as_name'";
my $sth_ex = $dbh->prepare($sql_ex);
$sth_ex->execute or &SendStatus("nfo", $J, "$trash$J$check_err", $MASTER_IP);
if($sth_ex->rows == 1){
        print "This machine is registered!\n".$sth_ex->rows."\n";
        return "here";
} else {
my $sql = "insert into $table (ip_as_name, port) values ('$ip_as_name', '$port')";
my $sth = $dbh->prepare($sql);
$sth->execute or &SendStatus( "nfo", $J, "$trash$J$add_err", $MASTER_IP) and return "error";
return "registered";
}
}
```

This one adds machine to database and checks if machine already stored in database.
And what is that misterious "SendStatus" do? Here you go:

```
sub SendStatus {
my($header, $splitter, $message, $ipaddress, $port) = @_; # Input params
socket(SOCKET, PF_INET, SOCK_DGRAM, $proto) or print "Cannot create socket!\n";
#^ socket creation
$ipaddr   = inet_aton($ipaddress); # ip addr
$portaddr = sockaddr_in($port, $ipaddr); # port and ip
my $msg = "$header$splitter$message";     # combined message
send(SOCKET, $msg, 0, $portaddr); # send data
shutdown(SOCKET, 0);            # I/we have stopped reading data
shutdown(SOCKET, 1);            # I/we have stopped writing data
shutdown(SOCKET, 2);            # I/we have stopped using this socket
print $msg."\n";
return;
}
```

By the way, i am sorry if going to fast but i am assuming that you have appropriate knowledge level to get the code on the fly and isnt considered about comments...
Are you ready for the last shot? After we'll catch some breath and discuss a couple of things.
Main function ahead, which does all the job. Receives from one machine and sends to another, processing data, anyways, welcome to the abyss ;]
Just joking :)

```
sub UdpListener{
my $response;
my $dec;
my $binded = 0;
my $trash = "Server~#";
my $rcli_sign = "R-Client~#";
my $rcli_here = "reg_ok";
my $rcli_regok = "registered";
my $xXx = $gpp1[rand(@gpp1)]." ".$gpp2[rand(@gpp2)]." ".$gpp3[rand(@gpp3)]."\n".$help;
my $check_err = "Cannot execute mysql query while checking machine($remoteaddress)!\n";
__init:
$response = "";
my $q=CGI->new();
socket(SOCKET, PF_INET, SOCK_DGRAM, $proto) or print "Cannot create listenning socket!\n";
setsockopt(SOCKET, SOL_SOCKET, SO_REUSEADDR, 1) ;

$ipaddr   = inet_aton($VMWareIp);
$portaddr = sockaddr_in($LISTEN_PORT, $ipaddr);
bind(SOCKET, $portaddr) or die "Could not bind! $!";
if($binded == 0){
	print "Binded\n";
} else {
	print "Rebinded\n";
}
my $remoteaddress = $q->remote_addr();
while(1){
$binded = 1;
$dec = "";
recv(SOCKET,$response,1000,0);
$dec = rot47($response);
print $dec."\n";
@words = split(/$J/, $dec);
#	print $words[0]."\n";
if($words[0] eq "?"){
	print "Help information sent\n";
	&SendStatus("nfo", $J, "$trash$J$xXx", $MASTER_IP, $SEND_PORT); # $whatever
	DestroySocket(SOCKET); goto __init;
	} elsif($words[0] eq "machine"){
	print $words[1]." ".$words[2]."\n";
	} elsif(index($words[0], "stats") > -1 || index($words[0], "statistics") > -1){
	&FetchMachines($MASTER_IP);
	DestroySocket(SOCKET); goto __init;
	} elsif($words[0] eq "die"){
	&SendStatus("nfo", $J, "$trash$J$death", $MASTER_IP);
	DestroySocket(SOCKET);
	system("clear");
	die "\nMaster ordered me to die ;(\n";
	} elsif($words[0] eq "rcli"){
		if($words[1] eq "status"){
		my $MACH_PORT = $words[2];
```

```
        $MACHINE_ADDR = $words[3];
        my $dbh = DBI->connect("dbi:mysql:$db:$dbhost",$dbuser,$dbpass) ; my $data;
        my $sql = "select * from $table where ip_as_name = '$MACHINE_ADDR'";
        my $sth = $dbh->prepare($sql);
        $sth->execute or &SendStatus ("nfo", $J, "$trash$J$check_err") and goto __init;
            if($sth->rows == 1){
            print "Machine ($MACHINE_ADDR) is here.\n";
            &SendStatus("", "", "$rcli_here", $MACHINE_ADDR, $MACH_PORT);
            DestroySocket(SOCKET); goto __init;
            } else {
            print "Machine ($MACHINE_ADDR) is not detected in our DB, adding...";
            my $status = AddMachine($MACHINE_ADDR, $MACH_PORT);
                if($status eq "registered"){
                &SendStatus("", "", "$rcli_regok", $MACHINE_ADDR, $MACH_PORT);
                DestroySocket(SOCKET); goto __init;
                } else {
                die "i dont want to check for any errors =/";
                }
            }
        } else {
            my $rcli_msg = $words[1];
            &SendStatus("nfo", $J, "$rcli_sign$J$rcli_msg\n", $MASTER_IP, $SEND_PORT);
            DestroySocket(SOCKET); goto __init;
        }
    } elsif($words[0] eq "2rcli"){
    my $MACHINE_ADDR = $words[1];
    my $rcli_msg = $words[2];
    &SendStatus("", "", "$rcli_msg", $MACHINE_ADDR, $MACHINE_PORT);
    DestroySocket(SOCKET); goto __init;
    } else {
    &SendStatus("nfo", $J, "$trash$J$whatever\n", $MASTER_IP, $SEND_PORT);
    DestroySocket(SOCKET); goto __init;
    }
}
}
```

Stop. Yeah, thats it ... for now ;)
This whole script uses 3 perl modules:

1. Socket (use Socket;)
2. DBI (use DBI;)
3. CGI (use CGI;)

Easy to notice, that socket module here is the most important and more often used. Assuming, that you are familiar with at least BSD sockets – there is nothing new. You may have some considerations about DBI module and what it does.

The DBI module enables your Perl applications to access multiple database types transparently. You can connect to MySQL, MSSQL, Oracle, Informix, Sybase, ODBC etc. without having to know the different underlying interfaces of each. The API defined by DBI will work on all these database types and many more.

You can connect to multiple databases of different types at the same time and easily move data between them. The DBI layer allows you to do this simply and powerfully.

*** from http://dbi.perl.org

In short, DBI provides support in your application for interactions with different types of databases. In our case – MYSQL Database. There are 5 main functions:

1. connect – connect to a database
2. prepare – prepare sql query
3. execute – execute sql query
4. disconnect – disconnect from database
5. do – a faster way to execute sql query when no data returned

Example:

```
my $dbh = DBI->connect(”dbi:mysql:$db:$dbhost”,$dbuser,$dbpass) ;
my $sq = ”select count(*) from $table”;
my $st = $dbh->prepare($sq);
$st->execute;
```

SITM server uses this module for adding machine in database and for keep it there, also for checking if it is there.

```
while (@row = $sth->fetchrow_array) {
$container .= $row[1].$online;
$container .= ”$J”;
}
```

This example processes returned data.

Another module is CGI.

CGI.pm is a stable, complete and mature solution for processing and preparing HTTP requests and responses. Major features including processing form submissions, file uploads, reading and writing cookies, query string generation and manipulation, and processing and preparing HTTP headers. Some HTML generation utilities are included as well.

CGI.pm performs very well in in a vanilla CGI.pm environment and also comes with built-in support for mod_perl and mod_perl2 as well as FastCGI.

It has the benefit of having developed and refined over 10 years with input from dozens of contributors and being deployed on thousands of websites. CGI.pm has been included in the Perl distribution since Perl 5.4, and has become a de-facto standard.

*** from perldoc.perl.org

In a manner of speaking, it is initialized here but not used in fact.

```
my $q=CGI->new();
my $remoteaddress = $q->remote_addr();
```

These 2 lines of code are responsible for retrieving ip address of connected remote machine,
but in case of our project – will do nothing because it will not fetch LAN ips or virtual subnet ips.
You can use CGI instead, when you are dealing with real servers and each got external ip address.

```
@words = split(/$J/, $dec);
```

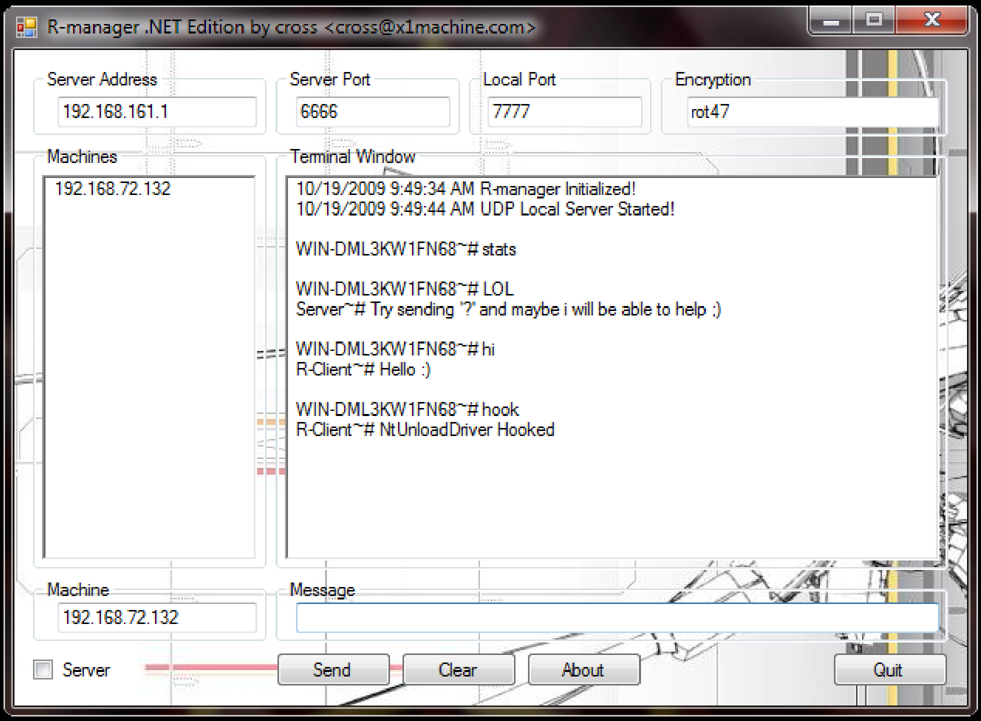
Here you can observe the fact i was talking about some time ago, about splitters and message parts.
Now we have tokens from @words array, and for example if the very first token equal rcli, script knows that it should send actual message to master IP. If token equal 2rcli – message should be sent to remote client, and so on.
Gosh, i need a smoke and coffee – i have been writing this paper for about 20 hours now, among with finishing and testing the project itself. In a meantime, enjoy the picture i've created just now :)

```
#undef ARTICLE_STATUS
        #define ARTICLE_STATUS "67%"
```

| R-manager |

Just wandering, how much time it will take yet and if i could make it at one session...
R-Manager – our monitor and control application, with graphical user interface, placed on our personal machine, Linux and Windows editions. Well, its time to lift the curtain on a little bit and present to you how they are looks like.

R-Manager C# .NET Windows Edition



Server Address: 192.168.161.1
Server Port: 6666
Local Port: 7777
Encryption: rot47

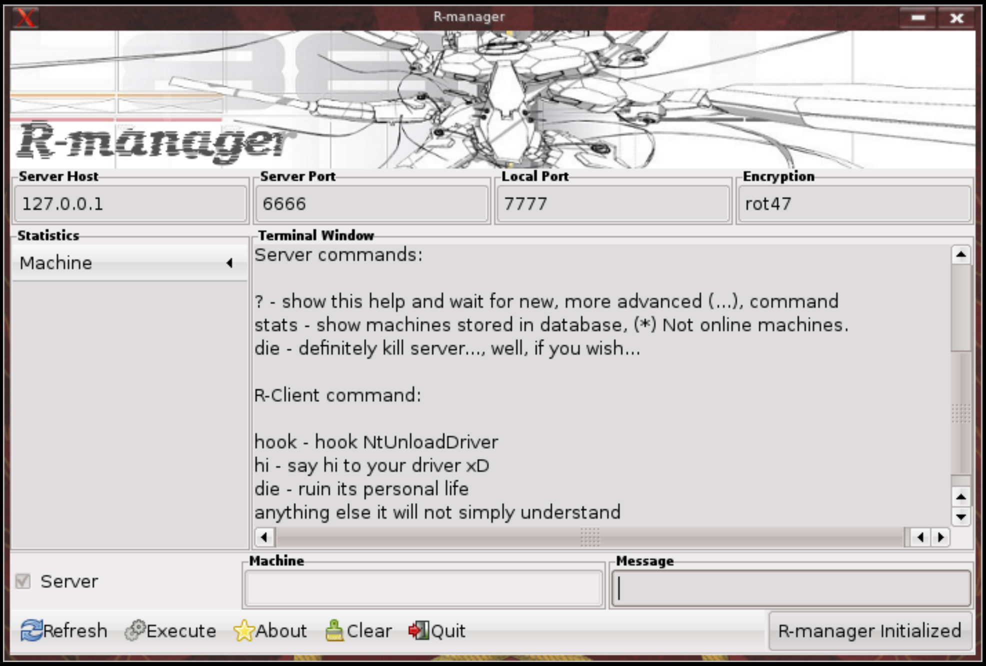
Machines: 192.168.72.132

Terminal Window:

```
10/19/2009 9:49:34 AM R-manager Initialized!
10/19/2009 9:49:44 AM UDP Local Server Started!

WIN-DML3KW1FN68~# stats

WIN-DML3KW1FN68~# LOL
Server~# Try sending '?' and maybe i will be able to help ;)

WIN-DML3KW1FN68~# hi
R-Client~# Hello :)

WIN-DML3KW1FN68~# hook
R-Client~# NtUnloadDriver Hooked
```

Machine: 192.168.72.132
Message:

R-Manager Gtk+ Linux Edition



Server Host: 127.0.0.1
Server Port: 6666
Local Port: 7777
Encryption: rot47

Statistics: Machine

Terminal Window:

```
Server commands:

? - show this help and wait for new, more advanced (...), command
stats - show machines stored in database, (*) Not online machines.
die - definitely kill server..., well, if you wish...

R-Client command:

hook - hook NtUnloadDriver
hi - say hi to your driver xD
die - ruin its personal life
anything else it will not simply understand
```

Machine:
Message:

Pretty simple interfaces with minimum of functionalities which makes them easier to understand for a novice and easier to customize for advanced programmer to fit ones needs. Well, i am not going to reinvent the wheel, describing specifics of each programming language i have used in these examples, but rather point to 2 major web resources where you can find complete documentation and learn from it.

http://msdn.microsoft.com
http://library.gnome.org/devel/gtk/2.16/

And yet again, the same networking design we have here: our application binds datagram socket, receives, and sends messages. At the screenshots above one element of graphical interface is missing. A networking initialization button. I left this up to you, when you will decide to launch it – just hit the button. In case of Gtk+ version – this button called "Network", in case of C# - Initialize (red colour); buttons will dissapear then. Lets take a look what happens.

Gtk+

```
int run_network_function(){
gtk_widget_hide(GTK_WIDGET(run_network)); // here we are hiding button
//======================================================
// Creating udp listener
pthread_t thread_id;          // pthread id
thdata thread_data;// pointer to thread structure
CLEAR(thread_data.local_port); // ZeroMemory
const char *LocalPort = gtk_entry_get_text(GTK_ENTRY(local_port_ent)); // get local port from entry
// widget, by the way, Gtk+ Entry = C# TextBox
strcpy(thread_data.local_port, LocalPort); // fill thread structure
pthread_create (&thread_id, NULL, &UdpListener, &thread_data); // create thread
//======================================================
}
```

C# .NET

```
   private void Button_Click(object sender, System.EventArgs e)
{
   Thread trd = new Thread(new ThreadStart(this.ThreadTask)); // initialize thread class
   trd.IsBackground = true;    // run in background
   trd.Start();     // start
   start_server.Hide(); // hide button
}
```

Lets take a closer look, what happens in each thread.

Gtk+

```
void * UdpListener(void *ptr){
thdata *data;
data = (thdata *) ptr;
char *t; int i;
int sock, sockh;
char message[10000];
char *token[256];
struct sockaddr_in our_addr, serv_addr;
socklen_t length;
char *machine_name;
int machines_count;
char xXx[1024];
char new_line[] = "\n";
bzero(&our_addr,sizeof(our_addr));
our_addr.sin_family = AF_INET;
our_addr.sin_addr.s_addr=htonl(INADDR_ANY);
our_addr.sin_port=htons(atoi(data->local_port));
sock=socket(AF_INET,SOCK_DGRAM,0);        // initialize Datagram socket
bind(sock, (struct sockaddr *)&our_addr, sizeof(our_addr)); // bind to local address
printf("udp server started!\n");
while(1){ // one big loop
length = sizeof(serv_addr);
sockh = recvfrom(sock,message,10000,0,(struct sockaddr *)&serv_addr,&length);
// receiving message from SITM server
message[sockh] = 0;
t = strtok(message, "!");
for(i = 0; t; t = strtok(NULL,"!"), i++) token[i] = t; // tokenizing incomming message
if(strcmp(token[0], "mcs") == 0){ // here we are getting machine ip
        machines_count = atoi(token[2]); // number of machines - ommited
        machine_name = token[3]; // name of the machine, its IP address in our case
        gtk_list_store_append (store_ex, &iter_ex); // append iter in "datagrid"
        gtk_list_store_set (store_ex, &iter_ex,      // update with new value
                  COLUMN_MACHINE, machine_name,
                  -1);
        } else if (strcmp(token[0], "nfo") == 0){ // standard message
        strcpy(xXx, token[1]); // prepare message
        strcat(xXx, " ");
        strcat(xXx, token[2]);
        Update_Buffer_And_Scroll_To_End(log_view, xXx); // Update terminal windows with new
// message
        }
        CLEAR(message); // ZeroMemory
        continue; // continue the game :)
        }
}
```

Old good UNIX sockets.

C# .NET

```
    private void ThreadTask()
    {
    int recv;
    String TimeAndDate = "";
    DateTime thisDate = DateTime.Now;
    TimeAndDate = String.Format("{0:G}", thisDate); // get local time
    byte[] data = new byte[1024];
    String LocalPort = local_port_ent.Text;   // get port from texbox
    int LoPo = System.Int32.Parse(LocalPort);         // pars port
    IPEndPoint ipep = new IPEndPoint(IPAddress.Any, LoPo); // init  IPEndPoint class

    Socket newsock = new Socket(AddressFamily.InterNetwork,
                    SocketType.Dgram, ProtocolType.Udp); // init socket class
    newsock.Bind(ipep);          // bind socket

    IPEndPoint sender = new IPEndPoint(IPAddress.Any, 0);
    EndPoint Remote = (EndPoint)(sender);
    termo_label.Text += TimeAndDate + " UDP Local Server Started!\n";
    while (true)
    {
        String response = "";
        recv = newsock.ReceiveFrom(data, ref Remote); // receive message
        System.Text.ASCIIEncoding enc = new System.Text.ASCIIEncoding();
        response = enc.GetString(data);
        Array.Clear(data, 0, data.Length); // then clear
        char[] separator = {'!'};
        string[] token = response.Split(separator); // "tokenize" message
    // and then processing data, like in Gtk+ example
        if (String.Compare(token[0], "mcs") == 0)
        {
            machine_name.Text = token[3];
        }
        else if (String.Compare(token[0], "nfo") == 0)
        {
            termo_label.Text += token[1] + " " + token[2];
        }
    // scroll terminal window down
        SendMessage(terminal_pannel.Handle, WM_VSCROLL, (IntPtr)SB_PAGEDOWN, IntPtr.Zero);
    }
}
```

So these threads processing incomming data and updating terminal window. Lets examine, what happens, when Send button is pressed. Notice: you dont need to press send button at all, you just can hit ENTER while typing in Message entry / textbox field.

Gtk+

```
int TransmitData(){
int result;
regex_t  rx;
regmatch_t   *matches;
char buf[] = "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.(25[0-5]|2[0-4]
[0-9]|[01]?[0-9][0-9]?)\\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)";
// ip address regular expression
bool Send2Machine;
char MainMessage[1024];
char no_serv_err[] = "\nServer IP not specified! Terminating task...\n";
printf("TransmitData called!\n");
CLEAR(MainMessage);
// getting needed values from Entry widgets
const char *MachineName = gtk_entry_get_text(GTK_ENTRY(machine_ent));
const char *MessageEnt = gtk_entry_get_text(GTK_ENTRY(message_ent));
const char *ServerAddr = gtk_entry_get_text(GTK_ENTRY(serv_host_ent));
const char *ServerPort = gtk_entry_get_text(GTK_ENTRY(serv_port_ent));
const char *LocalPort = gtk_entry_get_text(GTK_ENTRY(local_port_ent));
bool server_check_button_status =
gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(server_check_button));
// get server check button state
if(strcmp(MessageEnt, "clear") == 0){ // if our message is simply 'clear'
        clear_output();         // then clear terminal window without sending any data
        return 1;        // like in linux terminal
        }

if(strcmp(MessageEnt, "exit") == 0) exit(0); // same like above, in this case exit

result = regcomp( &rx, buf, REG_EXTENDED ); // compile regular expression pattern
matches = (regmatch_t *)malloc( (rx.re_nsub + 1) * sizeof(regmatch_t) ); // allocate memory

if(strcmp(ServerAddr, "") == 0){ // if no server address
Update_Buffer_And_Scroll_To_End(log_view, no_serv_err);
return 1;
}
result = regexec( &rx, ServerAddr, rx.re_nsub + 1, matches, 0 ); // validate server address
if (!result) { // if OK
printf("Valid IP!\n");
} else { // update terminal window about an error
Update_Buffer_And_Scroll_To_End(log_view, "Invalid server IP\n");
return 1;
}

if(strcmp(MachineName, "") != 0) { // if we have some value inside machine entry widget
Send2Machine = true; // we will send our message through SITM to machine
if(server_check_button_status == true){ // but if we using quick switch to server and it is active
        Send2Machine = false; // cancel
        goto next; // skip validation of machine's ip address
        }
        result = regexec( &rx, MachineName, rx.re_nsub + 1, matches, 0 ); // validate machine's ip
        if (!result) printf("Valid machine's IP!\n"); // good
         else { // bad
        Update_Buffer_And_Scroll_To_End(log_view, "Invalid Machine IP!\n");
        return 1;
        }
} else {
Send2Machine = false;
}
next:

        if(Send2Machine){ // if message is going to machine
        printf("sending data to machine!\n");
        strcpy(MainMessage, "2rcli!");
        strcat(MainMessage, MachineName);
        strcat(MainMessage, "!");
        strcat(MainMessage, MessageEnt);
        char *RottedMainMessage = code_rot_ex(MainMessage);
        UdpSender((char *)ServerAddr, (char *)ServerPort, RottedMainMessage);
        CLEAR(MainMessage);
        } else { // if message is only for server
        strcpy(MainMessage, MessageEnt);
        char *RottedMainMessage = code_rot_ex(MainMessage);
        printf("machine name is null - sending data to server!\n");
        UdpSender((char *)ServerAddr, (char *)ServerPort, RottedMainMessage);
        // call udp sender function
        CLEAR(MainMessage); // ZeroMemory
        }

}


int UdpSender(char *addr, char *port, char *message){
// basic things everyone should know
int sock, sockh;
char recv_msg[10000];
struct sockaddr_in serv_ddr;
bzero(&serv_ddr,sizeof(serv_ddr));
serv_ddr.sin_family = AF_INET;
serv_ddr.sin_addr.s_addr=inet_addr(addr);
serv_ddr.sin_port=htons(atoi(port));
sock = socket(AF_INET,SOCK_DGRAM,0);
    sendto(sock,message,strlen(message),0,
 (struct sockaddr *)&serv_ddr,sizeof(serv_ddr));
return 0;
}
```

Here we are doing exactly the same things like in previous functions, so will skip commenting

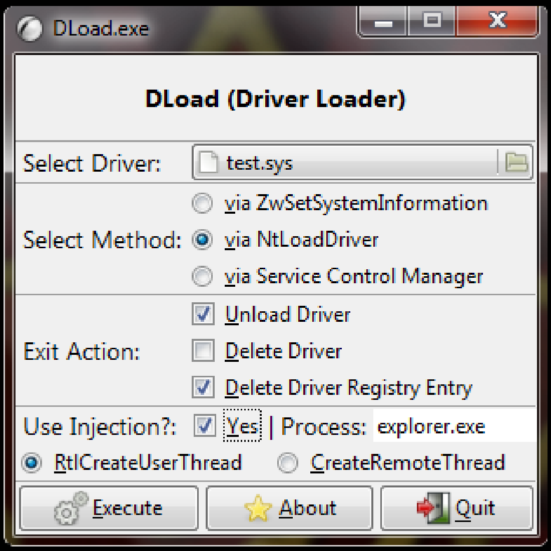
```
public bool IsValidIP(string addr)
{
    string pattern = @"\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b";
    Regex check = new Regex(pattern);
    bool valid = false;
    valid = check.IsMatch(addr, 0);
    return valid;
}
```

```
private void Send_Function(object sender, System.EventArgs e) {
    byte[] data = new byte[1024];
    bool SendToMachine = false;
    String DestMSG;
    String message;
    String dest_ip = serv_host_ent.Text;
    String MachineIp = machine_name_ent.Text;
    if (String.Compare(dest_ip, "") == 0)
    {
        termo_label.Text += "\nServer IP not specified! Terminating task...\n";
        return;
    }
    if (String.Compare(MachineIp, "") != 0)
    {
        SendToMachine = true;
        if (goto_server_hax.Checked)
        {
            SendToMachine = false;
            goto __next;
        }
        if (IsValidIP(MachineIp) == false)
        {
            termo_label.Text += "\nMachine IP is not valid! Terminating task...\n";
            return;
        }
    }

    __next:
    if (IsValidIP(dest_ip) == false)
    {
        termo_label.Text += "\nIP is not valid! Terminating task...\n";
        return;
    }
    IPEndPoint ipep = new IPEndPoint(
        IPAddress.Parse(dest_ip), 6666);
    Socket server = new Socket(AddressFamily.InterNetwork,
       SocketType.Dgram, ProtocolType.Udp);

    DestMSG = message_entry.Text;
    if (SendToMachine)
        message = "2rcli!" + MachineIp + "!" + DestMSG;
    else message = DestMSG;

    if (String.Compare(message, "") == 0) {
        termo_label.Text += "\nNo message! Terminating task...\n";
        return;
        }
    message_entry.Text = "";
    if (!SendToMachine)
        termo_label.Text += "\n" + Environment.MachineName + "~# " + message + "\n";
    else
        termo_label.Text += "\n" + Environment.MachineName + "~# " + DestMSG + "\n";

    if (String.Compare(DestMSG, "exit") == 0)
    {
        Quit_Function(sender, e);
        return;
    }
    if (String.Compare(DestMSG, "clear") == 0)
    {
        termo_label.Text = "";
        return;
    }

    data = Encoding.ASCII.GetBytes(Rot47c(message));
    server.SendTo(data, data.Length, SocketFlags.None, ipep);
    SendMessage(terminal_pannel.Handle, WM_VSCROLL, (IntPtr)SB_PAGEDOWN, IntPtr.Zero);
    return;
}
```

The rest of the code mainly regards graphical interface.

The last, almost not important thing, i would like to propose my simple driver loader for loading R-Client driver, that's of course if you don't have your own.

Well, it is written in Gtk+ (compiled for windows) and has following options:



DLoad (Driver Loader)

Select Driver: test.sys

Select Method:
○ via ZwSetSystemInformation
● via NtLoadDriver
○ via Service Control Manager

Exit Action:
☑ Unload Driver
☐ Delete Driver
☑ Delete Driver Registry Entry

Use Injection?: ☑ Yes | Process: explorer.exe

● RtlCreateUserThread ○ CreateRemoteThread

[+] Load driver with ZwSetSystemInformation
[+] Load driver with NtLoadDriver
[+] Load driver with Service Control Manager
[+] Unload driver
[+] Delete driver file
[+] Delete driver rigistry entries
[+] Injection of driver loading routine into another process
[+] Injection with CreateRemoteThread
[+] Injection with RtlCreateUserThread

Source and binary are included in project source.

```
#undef ARTICLE_STATUS
        #define ARTICLE_STATUS "98.5%"
```

| OUTRO |

To do for you my friend:

1. Configure R-client, R-manager, SITM server properly
2. Run SITM server
3. Run R-manager
4. Load R-client
5. Read the code
6. I could forgot about something and that's why bugs could appear (I am a human and make mistakes, besides while writing this I haven't slept for 24h)

This article was written for educational purposes and for fun. I haven't discussed here a lot of things which I should have mentioned,
but I have shown you the way it could be done. Lets say, I have started the subject ;)
For any questions, suggestions feel free to contact me <cross@x1machine.com>.
And who knows, maybe there will appear extended edition of this paper and of the whole project.
Best regards, always yours
cross.

```
#undef ARTICLE_STATUS
        #define ARTICLE_STATUS "100%"
```