Using ICMP tunneling to steal Internet

A short tutorial by Doug Lawrie, nulldigital.net

Introduction

The scenario is you are without Internet connectivity anywhere. You have found either an open wireless access pointed or perhaps you're staying in a hotel which permits rented Internet via services like Spectrum Interactive [1] (previously known as UKExplorer). You make the connection, whether its physically connecting the Ethernet cables, or instructing you're wireless adapter to lock onto the radio signal. You are prompted with some sort of authorization page when you open a browser. You don't have access to it, so what do you do?

Open a command prompt or terminal and ping an IP you know to be up, active and that you know to be not physically connected to the network you are jacking into. Perhaps even just ping google.com

```
[16:03] kay@client.example.com:~$ ping google.com
PING google.com (64.233.187.99) 56(84) bytes of data.
64 bytes from 64.233.187.99: icmp_seq=1 ttl=237 time=196 ms
64 bytes from 64.233.187.99: icmp_seq=2 ttl=237 time=164 ms
64 bytes from 64.233.187.99: icmp_seq=3 ttl=237 time=152 ms
64 bytes from 64.233.187.99: icmp_seq=4 ttl=237 time=153 ms
```

Right, so we're connected, and we can use ICMP ping requests. We are now in a position to “steal Internet”. Now I do not promote any sort of actual illegal activity, but since these networks are “open” to us and acting as a service provider freely with no warnings of what we're doing could be considered unauthorized usage. Since ICMP works we can only assume that we're permitted to use it right? I don't see why they would allow this if they considered it unauthorized without notifying us beforehand.

For those that don't actually know what all this mention of ICMP is, I suggest you google it, or check out the wikipedia article on ICMP [2]

Building the tunnel

To set up the tunnel you need:

- One up system with Internet connectivity that can receive ICMP traffic and make outbound TCP connections. This system we will call the proxy.example.com.
- You (preferably) hold root or administrator level access on the proxy system.
- You (preferably) hold root or administrator level access on your local system
- A copy of PingTunnel [3] by By Daniel Stødle

The first step is simply to install PingTunnel. This is a extremely easy installation.

```
[16:15] kay@proxy.example.com:~$ wget
http://www.cs.uit.no/~daniels/PingTunnel/PingTunnel-0.61.tar.gz
--16:15:49--  http://www.cs.uit.no/~daniels/PingTunnel/PingTunnel-0.61.tar.gz
           => `PingTunnel-0.61.tar.gz'
Resolving www.cs.uit.no... 129.242.16.40
Connecting to www.cs.uit.no|129.242.16.40|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 53,433 (52K) [application/x-gzip]

100%[====================================>] 53,433        14.12K/s    ETA 00:00

16:15:56 (14.11 KB/s) - `PingTunnel-0.61.tar.gz' saved [53433/53433]
```

```
[16:15] kay@proxy.example.com:~$ tar xzf PingTunnel-0.61.tar.gz
[16:16] kay@proxy.example.com:~$ cd PingTunnel
[16:16] kay@proxy.example.com:~/PingTunnel$ make
gcc -Wall -g -MM *.c > .depend
gcc -Wall -g -c -o ptunnel.o ptunnel.c
gcc -Wall -g -c -o md5.o md5.c
gcc -o ptunnel ptunnel.o md5.o -lpthread -lpcap
[16:16] kay@proxy.example.com:~/PingTunnel$
```

Do this on both the proxy system and your local system.

On windows there are some ported copies of PingTunnel which are unsupported by the original creator, but I have used them and can they worked fine for me. You can find them here. One of the packages I believe should be ready for compiling with Microsoft Visual C++, the other is certainly ready for compiling with the windows gcc.

You will need the pcap library [4] to compile PingTunnel. For instructions on the libpcap installation see the appropriate manual. If you don't feel like going through all the trouble of compiling this, there should be an executable binary within one of the packages precompiled that you can just run.

Making a tunnel

On the proxy system, obtain administrator level privileges, and run PingTunnel as below

```
[16:16] root@proxy.example.com:~/PingTunnel$ ./ptunnel -f ping.log &
[16:16] root@proxy.example.com:~/PingTunnel$
```

This will run ptunnel in proxy mode and will log it's output to a file called ping.log rather than print it on stdout.

On the client system, obtain administrator level privileges again and run PingTunnel as below, lets assume we wanted to ssh to one of our servers, lets call it portal.example.com.

```
[16:16] root@client.example.com:~/PingTunnel$ ./ptunnel -p proxy.example.com -lp 8765
-da portal.example.com -dp 22 &
[inf]: Starting ptunnel v 0.60.
[inf]: (c) 2004-2005 Daniel Stoedle, daniels@cs.uit.no
[inf]: Relaying packets from incoming TCP streams.
[1] 10858
[16:17] root@client.example.com:~/PingTunnel$
```

Traveling through the tunnel

We now have a tunnel set up. So lets use it! To tell programs to use our tunnel we simply tell them to connect to port 8765 on the localhost. So for example

```
[16:16] kay@client.example.com:~$ ssh localhost -p 8765
The authenticity of host 'localhost (127.0.0.1)' can't be established.
RSA key fingerprint is b7:2b:f3:3d:08:86:20:55:1d:82:08:2b:d0:33:fe:af.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
Linux portal.example.com 2.4.27-0.3um #1 Thu Sep 2 11:39:16 GMT 2004 i686
root@portal.example.com:~#
```

Hurray, light at the end of the tunnel. We've used ICMP to make a proxied TCP connection to the outside world. We can use the same principle for any TCP based service.

Tunnel authentication

Potential problem. Our proxy end of the tunnel is available to be used by absolutely anyone. It has no authentication mechanism to permit only us to use it for tunneling. After all we don't want to get busted for sending spam because some jerk used our tunnel for his own illegal activity. So we need to secure it. The PingTunnel program provides a simple authentication option that allows us to set a password for our tunnel. Its not solid security but it's enough to keep our the majority of people. Especially since if we did a standard scan on our proxy we wouldn't see the ICMP daemon anyway.

To set a password, when you run the proxy end, use the -x switch like this.

```
[16:16] root@proxy.example.com:~/PingTunnel$ ./ptunnel -x ubersecret -f ping.log
&
[16:16] root@proxy.example.com:~/PingTunnel$
```

Then when our client requests a tunnel we run it like so.

```
[16:16] root@client.example.com:~/PingTunnel$ ./ptunnel -x ubersecret -p
proxy.example.com -lp 8765 -da portal.example.com -dp 22 &
[inf]: Starting ptunnel v 0.60.
[inf]: (c) 2004-2005 Daniel Stoedle, daniels@cs.uit.no
[inf]: Relaying packets from incoming TCP streams.
[1] 10858
[16:17] root@client.example.com:~/PingTunnel$
```

There you have it. Free Internet, no authentication needed with the service provider you are jacking into provided as they permit ICMP traffic.

Limitations.

There is some limitations however, the functionality of web browsers is limited due to having to make several connections to different places for a single page. The HTML will not indicate it should be passed through localhost. In HTTP 1.1 the Host header must be passed, most clients will pass the host of whatever domain they are currently connecting to, so you will have requests for host: localhost, which most probably wont exist on the web servers configuration and will just be rejected.

How it works

Enough of all this tutorial-like talk! Let's explain how it works. First of its important to know that ICMP is part of the IP suite of protocols. It's also important to be aware that like UDP, ICMP works on a datagram basis with no sort of transmission control or insurance of delivery. Finally its very important to note that TCP does have transmission control and mechanisms to ensure delivery or be alerted to a failure.

Lets take a look at the ICMP packet layout for echo requests and echo replies.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-------------------------------+
|     Type      |     Code      |          Checksum             |
+-------------------------------+-------------------------------+
|           Identifier          |        Sequence Number        |
+-------------------------------+-------------------------------+
|                         Data ...
+----------------------------------------------
```

The type field specifics what the packet is for, since ICMP is used for more than simple ping and ping replies there are many values, but for us we only need to be aware of two types. Type 8 tells us the packet is an echo request. Type 0 tells us its a echo reply.

The code field is for more information about the nature of the packet, but for both echo reply and echo request the code field will always be 0. If you want to know more about what the code field is used for see the ICMP RFC [5]

The checksum is a 16-bit one's complement of the one's complement sum of the ICMP message beginning with the ICMP type.

The identifier is used when the code field is 0, for echo requests & replies. Its used to help match requests & replies. This is often some information about the session, perhaps a PID or similar.

The sequence number, like the identifier is used as a to track which reply matches up with which request by way of a numerical value. This should be unique to all outstanding ICMP traffic. The RFC does not explicitly specify this. However if the ICMP implementation has a ambiguous identifier field and duplicate sequence numbers in outstanding, traffic could end up in confusion.

The data field of the request contains data to be copied into the replies data field. This could be used to add yet another way to identify request session, but in the case of the PingTunnel, this is one of the most important fields. This is where we put our ping tunnel protocol data into, which will include the TCP data.

For an overview of the sequence of packets during standard operations such as pinging hosts, or traceroutes this pdf document [6] gives an excellent brief.

So thats the background knowledge. The client will always be using type 8 request packets for all its traffic, the proxy will always use type 0 reply packets for all its traffic. The reason for this is simple routing. Say that the client was behind a router the router's operating system may itself reply to the proxies echo requests without forwarding them to a host within its network that is involved in the ICMP tunnel session. This also brings up an important point about the initial setup of the ICMP proxy host. It must be able to receive echo requests itself. This might make the host system a router or simply have all packets for the proxy IP routed directly to it.

The ping tunnel protocol is similar to TCP [7], stuffed ICMP data field with an extra field included. The extra field is called by the creator of PingTunnel, the “magic number” in simple terms its just an identifier to separate the echo traffic from regular echo traffic. Since the proxy host operating system should always reply with a regular echo reply to each request packet we send.

The internals of the ping tunnel protocol (no byte scaling).

```
+----------------+------+--------+---------+------------------+
|  Magic Number  |  IP  |  Port  |  State  |   Acknowledgment  |
+---------+------+------+--+-----+------+--+------------------+
| Length  |  Sequence No.  |  Reserved  |        Data ...
+---------+----------------+------------+--------------------
```

The IP and port fields are purely used during the first echo request from the client to the proxy to setup where the proxy should forward received packets. These are both

32 bits wide.

The state field is crucial in the TCP-like control of the protocol. It shows what sort of semantics should be understood from the packet. Such as starting a session (state = 0), ending it (state = 3), authentication (state = 4), data forwarding (state = 1), and the TCP-like acknowledge data (state = 2). It also contains a flag indicating who send the packet, proxy or client. This is used with the magic number to differentiate tunnel ICMP traffic from the redundant echo replies that the proxy will make - since the proxy reply will contain identical data to the request, this will tell the client to discard packets claiming to be from the client, but are actually from the proxy. When the left most bit is set the proxy is sending the data, when the left most but one bit is set the client is sending the data. The state field is 32 bit wide. For example say state is 0x80000004 then we can see that the proxy is forward some received data back to the client.

The acknowledgment and sequence number fields are used to ensure delivery in the correct order and also to detect when a packet could get lost in the network cloud between the client and proxy. These are very much like the TCP fields of the same name, except the sequence number here is a 16 bit number thats allowed to wrap around just before overflow. The acknowledgment field is 32 bit wide, like it is in TCP. As the conversation progresses the sequence number is incremented both client and proxy recording the last received sequence number acknowledge by the remote peer. The acknowledgment timeout time used in ptunnel is 1500 milliseconds and after that time has elapsed without acknowledgment the packet is resent.

The length field is the number of bytes contained in the data field when the state field represents data forwarding (state = 1), and 0 otherwise. This is also a 32 bit wide field.

The reserved field is just for padding and is 16 bits wide.

The data field is exactly the same as the TCP data field and contains the actual data the be transported by TCP once the packet is received at the proxy.

For illustrated purposes, here is a diagram of the entire packet to be included within IP.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-------------------------------+
|     Type      |     Code      |          Checksum             |
+-------------------------------+-------------------------------+
|           Identifier          |        Sequence Number        |
+-------------------------------+-------------------------------+
|                          Magic Number                         |
+---------------------------------------------------------------+
|                              IP                               |
+---------------------------------------------------------------+
|                             Port                              |
+---------------------------------------------------------------+
|                             State                             |
+---------------------------------------------------------------+
|                      Acknowledgment Number                    |
+---------------------------------------------------------------+
|                             Length                            |
+-------------------------------+-------------------------------+
|        Sequence Number        |           Reserved            |
+-------------------------------+-------------------------------+
|                            Data ...
+-----------------------------------------------------
```

As you can see the ptunnel protocol adds a lot of extra data to simple ICMP packets.

Ping tunnel also has some neat mechanisms for controlling multiple connections using the identifier field discussed earlier. It also emulated TCP transmission “windows” where the two hosts take turns in sending data. See the TCP RFC [7] for more information about TCP transmission windows. The authentication mechanism is simple and effective by sending a challenge for which the client should take the md5 hash of the password, appending to the challenge, and Finally taking another md5 hash of that. The proxy when match up this simple authentication and will respond accordingly.

The light at the end

When using PingTunnel you will probably have to expect a bit of latency due to all the extra data going around, not to mention that you're having to go through a proxy to get out onto the Internet! I do wonder if two sequence numbers are absolutely necessary, but hey, it's only 2 bytes wide anyway.

PingTunnel is an awesome utility and well work having around, it could do with some development. I'm tempted to request to join the project because I want to work on building a proxy-like interface to the program, such that you set a program, such as a web browser to use a SOCKS or HTTP proxy on localhost at the user-defined port number. This will overcome the program with web browsers and allow regular HTTP traffic to function perfectly.

Another idea and expansion I'd love to bring to the project would be a TCP wrapper shell, such that once within the wrapper all TCP traffic is automatically caught and redirected through the tunnel on the fly, this is a big project and will lead to the possibility of simply being able to start up the system in the TCP wrapper shell and have the full Internet experience without even having to move a figure or mess about changing tunnel destinations and so forth.

References (Viewed 10th may 2006)
[1] http://www.spectruminteractive.co.uk/
[2] http://en.wikipedia.org/wiki/Icmp
[3] http://www.cs.uit.no/~daniels/PingTunnel/
[4] http://sourceforge.net/projects/libpcap/
[5] http://www.ietf.org/rfc/rfc792.txt
[6] http://www.eventhelix.com/RealtimeMantra/Networking/Icmp.pdf
[7] http://www.ietf.org/rfc/rfc793.txt