Address Spoofing with iptables in Linux

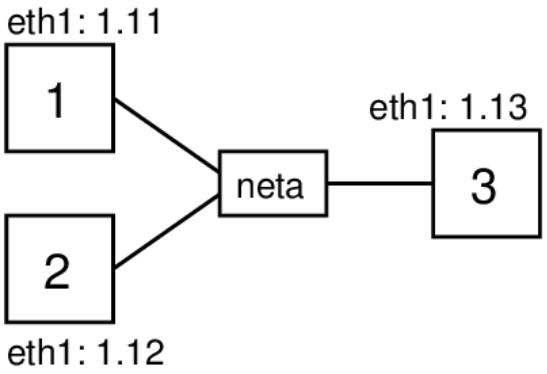
By Steven Gordon on Sun, 15/09/2013 - 12:34pm

Every IP datagram sent in the Internet contains a source and destination IP address in its header. The source is the original sender of the datagram and the destination is the intended recipient. So, ignoring the role of NAT [3], when your computer contacts a server on the Internet, that server knows your IP address as it is included in the source field of the IP datagram. In some cases you may want to change the source IP address included in the IP datagram (without changing your actual computer IP address). For example, this can be useful for network testing and diagnostics, security penetration testing and performing security attacks (for learning purposes only, of course). Setting the IP source address of datagrams to be a fake address is called *address spoofing*. In Linux it is very easy to do using `iptables`.

Address spoofing can be performed with a single command using `iptables`. For example, to change the source address included in IP datagrams that your computer sends to `1.1.1.1`:

```
$ sudo iptables -t nat -A POSTROUTING -j SNAT --to-source 1.1.1.1
```

Now lets see IP address spoofing in use with some examples using `ping`. I've setup a simple virtual network [4] consisting of three nodes on the same LAN as illustrated below. All IP addresses start with `192.168.`, so for example `node1` has address `192.168.1.11`.



Network topology for testing address spoofing

1. Ping Without Address Spoofing

First lets see what happens when one node communicates with another without address spoofing. The `ping` command triggers ICMP Echo Request packets to be sent to the destination IP address every one second. When a computer receives an ICMP Echo Request it will reply with a ICMP Echo Reply. On `node2`, which will be the destination, I will capture packets using `tcpdump`, and then on `node1` I will start `ping` using the `-c 2` option to limit the number of ICMP Echo Requests sent to 2:

```
network@node1:~$ ping -c 2 192.168.1.12
```

The captured packets on `node2` illustrate how `ping`/ICMP and ARP work.

```
network@node2:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
```

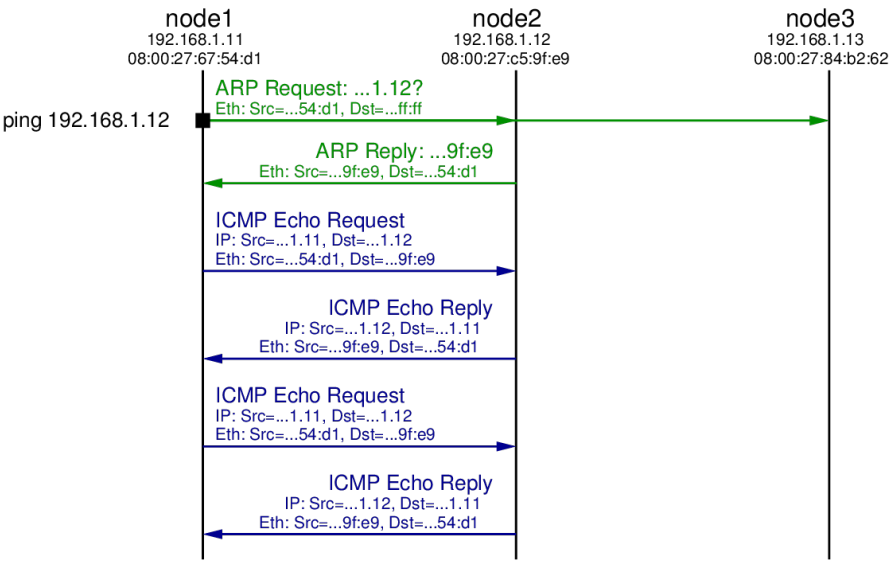
```
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
15:22:15.414345 ARP, Request who-has 192.168.1.12 tell 192.168.1.11, length 28
15:22:15.414364 ARP, Reply 192.168.1.12 is-at 08:00:27:c5:9f:e9 (oui Unknown), length 28
15:22:15.415085 IP 192.168.1.11 > 192.168.1.12: ICMP echo request, id 1072, seq 1, length 64
15:22:15.415102 IP 192.168.1.12 > 192.168.1.11: ICMP echo reply, id 1072, seq 1, length 64
15:22:16.415602 IP 192.168.1.11 > 192.168.1.12: ICMP echo request, id 1072, seq 2, length 64
15:22:16.415627 IP 192.168.1.12 > 192.168.1.11: ICMP echo reply, id 1072, seq 2, length 64
```

Before the first ICMP Echo Request packet is sent by `node1`, it must first discover the hardware address for the node with IP address `192.168.1.12`. Recall that all communications in a LAN are performed using the data link layer protocol, in this case Ethernet. Although `node1` knows the destination IP address, it must know the destination hardware (Ethernet or MAC) address to send the Ethernet frame to `node2`. The Address Resolution Protocol [5] (ARP) is used to perform this mapping of IP address to hardware address. `node1` broadcasts an ARP Request message to all nodes in the LAN, asking other nodes who has (knows) the hardware address for `192.168.1.12`. `node2` has this IP address, and therefore responds with an ARP Reply telling `node1` the corresponding hardware address: `08:00:27:c5:9f:e9`. Now `node1` can send the ICMP Echo Request to `node2`.

When `node2` receives the ICMP Echo Request and must reply with a ICMP Echo Reply. Note that `node2` already knows the hardware address of `node1` (it was cached from the ARP Request sent previously) and so an ARP Request is not needed. When the ICMP Echo Reply is received by `node1`, it records the round-trip time (RTT) from when `ping` was initiated until when the first ICMP Echo Reply is received. Note that this includes the time for the ARP Request and Reply.

A 2nd ICMP Echo Request is sent by `node1` about 1 second after the 1st. Another ICMP Echo Reply is eventually received, and then `ping` reports summary statistics of the RTT for both requests (not shown).

The packets sent in the LAN are illustrated in the figure below. Note that although the `ping` is between `node1` and `node2`, the ARP Request is broadcast using the special destination hardware address `ff:ff:ff:ff:ff:ff`. Therefore `node3` also receives the ARP Request. It does not however reply because it does not have the queried IP address `192.168.1.12`.



Ping Exchange: Normal Case

2. Fake Source Address is Non-Existent Node

Now lets repeat the `ping` from `node1` to `node2`, but set a fake source address in the ICMP Echo Requests

sent by `node1`. In this case the fake address will be a non-existent address: no other node in the LAN has the fake address. I'll choose `192.168.1.66`.

`iptables` is the packet filtering and firewall applications on Linux. It allows users to manipulate how the Linux kernel processes packets. I will not attempt to explain `iptables` in full here, instead focus just on how it can be used to set a spoofed source address. The command to use is:

```
network@node1:~$ sudo iptables -t nat -A POSTROUTING -p icmp -j SNAT --to-source 192.168.1.66
```

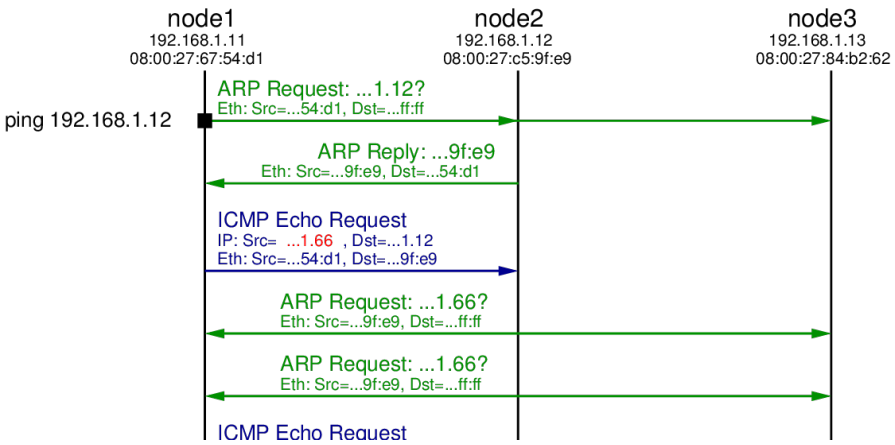
You must have appropriate privileges to use `iptables`, hence I preceded it with `sudo` to execute it as super user. The option `-t nat` indicates we want to manipulate that table used for Network Address Translation (NAT), i.e. translating one address to another. The `-A POSTROUTING` option indicates we want to add a rule to the POSTROUTING chain. The rules in this chain are applied to all packets which have completed the routing procedure on this computer and are about to be sent. That is, it is applied to packets just before they are sent by the hardware. The `-p icmp` option is a condition for the rule: packets must be using the protocol ICMP (this rule will not apply to non-ICMP packets). Next is the action to take when the rule is matched. `-j SNAT` means the action is to perform source network address translation, i.e. change the source address. Finally, the `--to-source 192.168.1.66` option indicates the new (spoofed) IP source address of the packet. In summary, all ICMP packets sent by `node1` will have source address `192.168.1.66`.

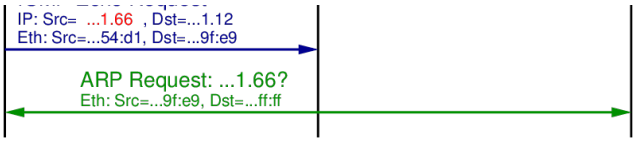
Now lets repeat the steps of pinging from `node1` to `node2`, while also capturing on `node2`. The output of `ping` and `tcpdump`, as well as a figure illustrating the packet exchange, follow.

```
network@node1:~$ ping -c 2 192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.

--- 192.168.1.12 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1000ms
```

```
network@node2:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
15:39:04.911564 ARP, Request who-has 192.168.1.12 tell 192.168.1.11, length 28
15:39:04.911600 ARP, Reply 192.168.1.12 is-at 08:00:27:c5:9f:e9 (oui Unknown), length 28
15:39:04.912523 IP 192.168.1.66 > 192.168.1.12: ICMP echo request, id 1089, seq 1, length 64
15:39:04.912550 ARP, Request who-has 192.168.1.66 tell 192.168.1.12, length 28
15:39:05.909310 ARP, Request who-has 192.168.1.66 tell 192.168.1.12, length 28
15:39:05.910691 IP 192.168.1.66 > 192.168.1.12: ICMP echo request, id 1089, seq 2, length 64
15:39:06.909323 ARP, Request who-has 192.168.1.66 tell 192.168.1.12, length 28
```





Ping Exchange: Ping Exchange: Fake source of 192.168.1.66

As in the original case, an ARP Request is broadcast to the lane by `node1` to find the hardware address of `192.168.1.12`. A reply comes from `node2`, which is followed by `node1` sending the 1st ICMP Echo Request. But note that the source IP address in the ICMP Echo Request is `192.168.1.66`, not `node1`'s real IP address (`192.168.1.11`). When `node2` receives this ICMP Echo Request it will reply to node that sent it with a ICMP Echo Reply. From `node2`'s perspective, the node that sent the request is computer with IP `192.168.1.66`. Since it hasn't communicate with anyone with this IP in the past, `node2` initiates ARP to find the corresponding hardware address for IP `192.168.1.66`.

We see an ARP Request broadcast to the LAN by `node2`. But since there is no node with IP address `192.168.1.66`, there will be no ARP Reply. After some time `node2` tries the broadcast ARP Reply again, but still no reply. Hence `node2` will not send a ICMP Echo Reply because it doesn't know the hardware address to sent it to.

Eventually `node1` sends the 2nd ICMP Echo Request, but again there will be no reply. After waiting for some time, the `ping` application on `node1` gives up and reports the summary statistics: 2 packets transmitted, 0 (reply) packets received.

3. Fake Source Address is Another Node on LAN

Now lets try again with a fake source address, but this time it is set to be the same as another node in the LAN, e.g. `192.168.1.13`. To change the source address I will delete the old rule (with the address `192.168.1.66` in `iptables` and then add a new rule with source address `192.168.1.13`:

```
network@node1:~$ sudo iptables -t nat -D POSTROUTING -p icmp -j SNAT --to-source 192.168.1.66
network@node1:~$ sudo iptables -t nat -A POSTROUTING -p icmp -j SNAT --to-source 192.168.1.13
```

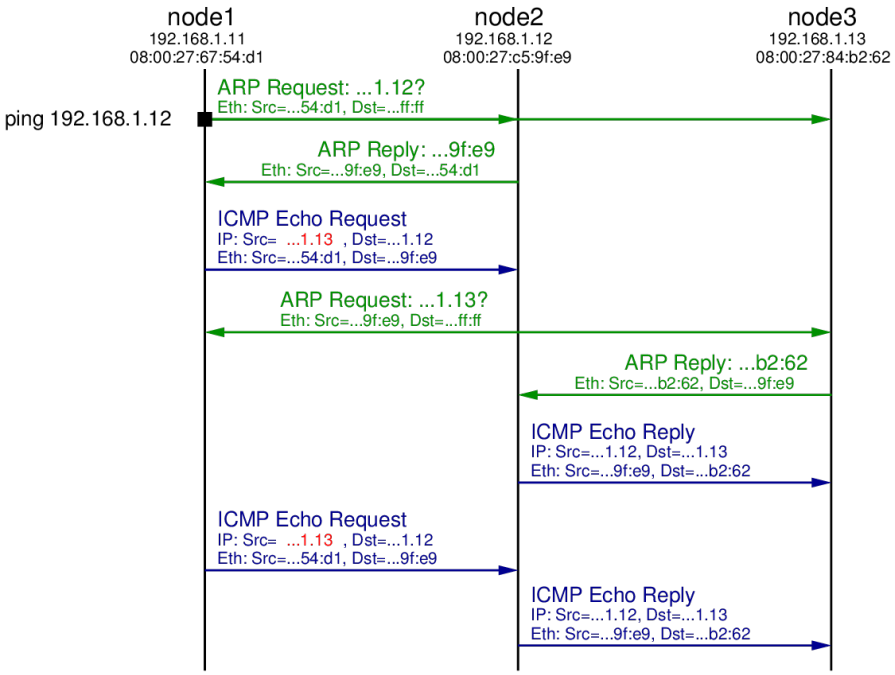
This time I will capture on both `node2` and `node3`. The output of `ping`, the two captures, and a diagram showing the packet exchange, follow.

```
network@node1:~$ ping -c 2 192.168.1.12
PING 192.168.1.12 (192.168.1.12) 56(84) bytes of data.

--- 192.168.1.12 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1008ms
```

```
network@node2:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
15:43:18.726532 ARP, Request who-has 192.168.1.12 tell 192.168.1.11, length 28
15:43:18.726565 ARP, Reply 192.168.1.12 is-at 08:00:27:c5:9f:e9 (oui Unknown), length 28
15:43:18.727435 IP 192.168.1.13 > 192.168.1.12: ICMP echo request, id 1098, seq 1, length 64
15:43:18.727463 ARP, Request who-has 192.168.1.13 tell 192.168.1.12, length 28
15:43:18.728149 ARP, Reply 192.168.1.13 is-at 08:00:27:84:b2:62 (oui Unknown), length 28
15:43:18.728156 IP 192.168.1.12 > 192.168.1.13: ICMP echo reply, id 1098, seq 1, length 64
15:43:19.734680 IP 192.168.1.13 > 192.168.1.12: ICMP echo request, id 1098, seq 2, length 64
15:43:19.734706 IP 192.168.1.12 > 192.168.1.13: ICMP echo reply, id 1098, seq 2, length 64
```

```
network@node3:~$ sudo tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
15:43:17.655046 ARP, Request who-has 192.168.1.12 tell 192.168.1.11, length 28
15:43:17.656292 ARP, Request who-has 192.168.1.13 tell 192.168.1.12, length 28
15:43:17.656310 ARP, Reply 192.168.1.13 is-at 08:00:27:84:b2:62 (oui Unknown), length 28
15:43:17.657025 IP 192.168.1.12 > 192.168.1.13: ICMP echo reply, id 1098, seq 1, length 64
15:43:18.663565 IP 192.168.1.12 > 192.168.1.13: ICMP echo reply, id 1098, seq 2, length 64
```



Ping Exchange: Ping Exchange: Fake source of 192.168.1.13

The initial ARP Request, ARP Reply and ICMP Echo Request are the same as in the previous test, except the source address in the ICMP Echo Request is the fake address `192.168.1.13`. When `node2` receives this ICMP Echo Request it determines it needs to send a reply to `192.168.1.13`. Therefore it initiates ARP, broadcasting an ARP Request in search for the hardware address of computer with IP `192.168.1.13`. That ARP Request is received by all nodes in the LAN, including `node3`. Since `node3` has the IP address `192.168.1.13` it replies with an ARP Reply indicating its own hardware address (`08:00:27:84:b2:62`).

When `node2` receives the ARP Reply it now knows the mapping of IP address `192.168.1.13` to hardware address `08:00:27:84:b2:62`. Hence it sends the ICMP Echo Reply to the computer that it thinks sent the ICMP Echo Request: `192.168.1.13`. So we see the ICMP Echo Request being received by `node3`.

In summary, `node1` initiated the `ping` to `node2`, but used a spoofed source address such that `node2` replied to `node3`. This concept can be used in a distributed denial of service attack: a malicious node initiates many pings to many nodes on the Internet, all with the spoofed IP address of some target node. All the nodes that receive the ICMP Echo Request send the ICMP Echo Reply to the one target node. With enough intermediate nodes, and repeating the process, the network to the target node can be overloaded, denying normal users access to the target node.

We have seen that it is very easy to use a fake (spoofed) IP address by using `iptables` in Linux. Its also

possible in other operating systems. Therefore when designing secure systems, using just a source IP address to identify nodes/users in a network is troublesome: anyone can easily use a fake address. Other techniques are needed for authentication, and also for limiting packets through the Internet that come from fake addresses.

Interest: Ubuntu Linux [6]
iptables [7]
Topic: Security [8]
Content: Howto [9]

Source URL: http://sandilands.info/sgordon/address-spoofing-with-iptables-in-linux

Links:
[1] http://sandilands.info/sgordon/address-spoofing-with-iptables-in-linux
[2] http://sandilands.info/sgordon/user/2
[3] http://en.wikipedia.org/wiki/Network_address_translation
[4] http://sandilands.info/sgordon/creating-a-virtual-network-of-linux-guests-using-virtualbox
[5] http://en.wikipedia.org/wiki/Address_Resolution_Protocol
[6] http://sandilands.info/sgordon/taxonomy/term/302
[7] http://sandilands.info/sgordon/taxonomy/term/311
[8] http://sandilands.info/sgordon/taxonomy/term/116
[9] http://sandilands.info/sgordon/taxonomy/term/212