**Learning Objectives**

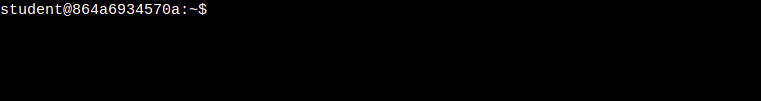
* Getting familiar with networking operations on Linux
* Understanding basic DNS and DHCP configuration settings
* Managing a DNS and DHCP service with dnsmasq

**Introduction**

Managing DNS and DHCP are very common tasks for system administrators. In large enterprises, this can be done using complex software and might even be done by separate teams. In smaller networks, you might be the only person behind all networking services and in a case like this, using a simple solution like dnsmasq might be appropriate

**Todo**

modify existing dnsmasq configuration to adapt it to the DNS and DHCP needs of your organization and then proceed to verify



**Linux commands reminder**

In this lab, we'll use a number of Linux commands that were explained during Course 3. This is a reminder of what these commands do:

* sudo <command>: executes a command with administrator rights
* ls <directory>: lists the files in a directory
* cat <file>: prints the whole contents of a file
* tail <file>: prints the end of a file (10 lines by default)
* nano <file>: opens a text editor to edit the file

We will also be using the service command, shown in a previous lab, and we will present a number of new commands like ip, dig and dhclient. We will briefly explain what these commands do when they are shown. Remember that you can always read the manual page using man <command\_name> to learn more about a command.

While you can copy and paste the commands presented in this lab, we recommend typing them out manually, to help with understanding and remembering these commands.

**The scenario**

The company that you are working for has dnsmasq set up to manage the DNS and DHCP needs of the network.

Currently, almost all of the DHCP range is being used to serve dynamic IPs. A number of servers will be added to the network and those should be configured with known IPs.

Your task in this lab is to modify the configuration of this dnsmasq setup so that those servers always have the same IPs. To do this, not only will you need to give those servers the requested IPs, but also reduce the range for the dynamic IPs.

**Network setup**

Because you follow the rule of never testing in production, you will experiment with the necessary changes in a machine that is simulating the network. You should do this instead of trying these commands out in the actual DNS server.

In real life, after you were done experimenting you would then apply the same changes that you tried in the test instance to the production instance that is running dnsmasq on the network.

Let's have a look at this simulated network configuration.

For our test instance, we have configured a virtual network interface (called eth\_srv) which the DNS and DHCP server will listen on. We can see the state of that interface with the ip command. The following will show information about the network configuration:

ip address show eth\_srv

Copied!

content\_copy

3: eth\_srv@eth\_cli: mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether 76:67:c9:a1:59:55 brd ff:ff:ff:ff:ff:ff

inet 192.168.1.1/24 scope global eth\_srv

valid\_lft forever preferred\_lft forever

We can see that this interface is configured to have the 192.168.1.1 IPv4 address in a network with a /24 or 255.255.255.0 netmask.

Additionally, we have another virtual network interface that we will use to simulate a client talking to the server and requesting DNS or DHCP traffic. This interface is called eth\_cli and we can see the state using an equivalent command as the one mentioned above:

ip address show eth\_cli

Copied!

content\_copy

2: eth\_cli@eth\_srv: mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether aa:fd:16:47:4a:cf brd ff:ff:ff:ff:ff:ff

In this case we can see that the interface does not have an IPv4 address (yet).

**Current configuration setup**

As mentioned, your company already has dnsmasq deployed on the network. In this lab we are going to make a few changes to the existing config. Let's look first at the current config.

The current config is stored in /etc/dnsmasq.d/mycompany.conf. Let's check out its contents with the cat command:

cat /etc/dnsmasq.d/mycompany.conf

Copied!

content\_copy

# This is the interface on which the DHCP server will be listening to.

interface=eth\_srv

# This tells this dnsmasq to only operate on that interface and not operate

# on any other interfaces, so that it doesn't interfere with other running

# dnsmasq processes.

bind-interfaces

# Domain name that will be sent to the DHCP clients

domain=mycompany.local

# Default gateway that will be sent to the DHCP clients

dhcp-option=option:router,192.168.1.1

# DNS servers to announce to the DHCP clients

dhcp-option=option:dns-server,192.168.1.1

# Dynamic range of IPs to use for DHCP and the lease time.

dhcp-range=192.168.1.2,192.168.1.254,24h

**Explanation of the meanings of these settings:**

* **interface** is the name of the interface that will be used for listening to DHCP requests and serving replies; we are setting it to our virtual eth\_srv interface.
* **bind-interfaces** means that dnsmasq will operate on that interface only and ignore the others.
* **domain** is the domain name used on the network.
* **dhcp-option** allows us to give the DHCP clients optional additional information. In this case we are setting the **router** (also known as default gateway) and the **dns-server**. When the clients receive the DHCP reply they will also receive and apply this configuration.
* **dhcp-range** indicates the range of IPs that are available to be used for dynamic IP assignment as well as the length of the lease. In this case, the whole network (except for the DHCP server) is currently available as part of the dynamic range. We know that we need to change this. The lease is currently set to 24h, which might not be a great idea if our network has a lot of devices that are only visible for short periods of time.

**Enabling debug logging**

When experimenting with changes in a service, it's a good idea to enable debug logging so that we can understand what's going on and why.

dnsmasq is running as a daemon in the background. We can query the status using the service command learned in the previous lesson:

sudo service dnsmasq status

Copied!

content\_copy

Checking DNS forwarder and DHCP server: dnsmasq(running).

We can see that the service is running and that the last lines were generated by it.

Note:If you get a prompt of the format lines 1-18/18 (END), this means you can scroll to the right to see any additional output that is currently not visible. You can quit this view by pressing q.

In order to enable debug logging we are going to stop the running service and modify the dnsmasq config file. To do that, first stop the dnsmasq daemon:

sudo service dnsmasq stop

Copied!

content\_copy

To edit the configuration, let's open the configuration file using a text editor called *nano*.

sudo nano /etc/dnsmasq.d/mycompany.conf

Copied!

content\_copy

In this file, we need to add the log-queries option, and tell dnsmasq where to log.

* At the end of this file, add a line that contains the log-queries option.
* After that line, add a line that defines the log file location, like this:

# This is the interface on which the DHCP server will be listening to.

interface=eth\_srv

# This tells this dnsmasq to only operate on that interface and not operate

# on any other interfaces, so that it doesn't interfere with other running

# dnsmasq processes.

bind-interfaces

# Domain name that will be sent to the DHCP clients

domain=mycompany.local

# Default gateway that will be sent to the DHCP clients

dhcp-option=option:router,192.168.1.1

# DNS servers to announce to the DHCP clients

dhcp-option=option:dns-server,192.168.1.1

# Dynamic range of IPs to use for DHCP and the lease time.

dhcp-range=192.168.1.2,192.168.1.254,24h

log-queries

log-facility=/var/log/dnsmasq.log

Once you've done this, press "Ctrl-X" to exit the editor. It will ask you to save your changes. Press "Y" for yes and then enter at the filename prompt.

We have made a change to the configuration file. Now we need to verify that it does what we intend it to do. The first step is to verify the syntax of the file is correct. To do this, we can use a parameter to dnsmasq: --test

sudo dnsmasq --test -C /etc/dnsmasq.d/mycompany.conf

Copied!

content\_copy

dnsmasq: syntax check OK.

If instead you get an error like: "dnsmasq: bad hex constant at line 21 of /etc/dnsmasq.d/mycompany.conf", it means that you have an error in the line indicated. Open the file with nano once again, and review and fix whatever is wrong in that line.

Once the configuration file passes a syntax check, start the daemon:

sudo service dnsmasq start

Copied!

content\_copy

Click Check my progress to verify the objective.

Enable debug logging

Check my progress

**Experimenting with DNS queries**

In order to check that this is working properly, let's start by asking some simple DNS queries and seeing what dnsmasq does with them. To do this, we will use the command dig, which allows us to request the IP address of a certain hostname. Let's try getting the IP address of example.com:

dig example.com @localhost

Copied!

content\_copy

; <<>> DiG 9.11.5-P4-5.1+deb10u5-Debian <<>> example.com @localhost

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8195

;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:

; EDNS: version: 0, flags:; udp: 512

;; QUESTION SECTION:

;example.com. IN A

;; ANSWER SECTION:

example.com. 21599 IN A 93.184.216.34

;; Query time: 3 msec

;; SERVER: 127.0.0.1#53(127.0.0.1)

;; WHEN: Wed Aug 11 10:52:25 UTC 2021

;; MSG SIZE rcvd: 56

By using the @localhost parameter, we told dig that we wanted to use the running machine as the DNS server. In other words, we queried our running dnsmasq and we got a reply from it. The reply contains the IPv4 address for the domain we requested.

Now, let's look at the debug logs written by dnsmasq, so you can see what dnsmasq said about the query:

sudo tail /var/log/dnsmasq.log

Copied!

content\_copy

Aug 11 10:52:25 dnsmasq[1217]: query[A] example.com from 127.0.0.1

Aug 11 10:52:25 dnsmasq[1217]: forwarded example.com to 169.254.169.254

Aug 11 10:52:25 dnsmasq[1217]: reply example.com is 93.184.216.34

This is telling us that it forwarded the request to another DNS server. This is the normal behavior for a caching DNS service like dnsmasq. Let's see what happens if we make the same request again. Run the same dig command. Hint: You can press the up-arrow to avoid having to type it again.

dig example.com @localhost

Copied!

content\_copy

; <<>> DiG 9.11.5-P4-5.1+deb10u5-Debian <<>> example.com @localhost

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17307

;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:

; EDNS: version: 0, flags:; udp: 4096

;; QUESTION SECTION:

;example.com. IN A

;; ANSWER SECTION:

example.com. 21482 IN A 93.184.216.34

;; Query time: 0 msec

;; SERVER: 127.0.0.1#53(127.0.0.1)

;; WHEN: Wed Aug 11 10:54:22 UTC 2021

;; MSG SIZE rcvd: 56

We got the same reply. What did dnsmasq say?

sudo tail /var/log/dnsmasq.log

Copied!

content\_copy

Aug 11 10:54:22 dnsmasq[1217]: query[A] example.com from 127.0.0.1

Aug 11 10:54:22 dnsmasq[1217]: cached example.com is 93.184.216.34

Notice how instead of forwarding the query to the nameserver of the machine, it has used the value that it had already cached.

Let's see what happens if we request an IP address of a domain name that doesn't exist.

dig example.local @localhost

Copied!

content\_copy

; <<>> DiG 9.11.5-P4-5.1+deb10u5-Debian <<>> example.local @localhost

;; global options: +cmd

;; Got answer:

;; WARNING: .local is reserved for Multicast DNS

;; You are currently testing what happens when an mDNS query is leaked to DNS

;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 22728

;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:

; EDNS: version: 0, flags:; udp: 512

;; QUESTION SECTION:

;example.local. IN A

;; AUTHORITY SECTION:

. 86399 IN SOA a.root-servers.net. nstld.verisign-grs.com. 2021081100 1800 900 604800 86400

;; Query time: 30 msec

;; SERVER: 127.0.0.1#53(127.0.0.1)

;; WHEN: Wed Aug 11 10:55:27 UTC 2021

;; MSG SIZE rcvd: 117

The reply we get indicates that the name doesn't have an associated IP address. Going back to the dnsmasq log, we can see what it said.

Aug 11 10:55:27 dnsmasq[1217]: query[A] example.local from 127.0.0.1

Aug 11 10:55:27 dnsmasq[1217]: forwarded example.local to 169.254.169.254

Aug 11 10:55:27 dnsmasq[1217]: reply example.local is NXDOMAIN

Because dnsmasq didn't have any information associated with that domain, it forwarded the query to the configured DNS server. The reply was that the domain didn't exist.

You can keep experimenting, querying for other domain names that you are interested in.

**Experimenting with a DHCP client**

Now that we know that DNS is working properly, let's experiment with the DHCP configuration. In the terminal, we will run dhclient, which is the most common DHCP client on Linux. As mentioned, we will run this on the eth\_cli interface. Additionally, we will tell it to run in verbose mode and run a debugging script that we provide:

sudo dhclient -i eth\_cli -v -sf /root/debug\_dhcp.sh

Copied!

content\_copy

Internet Systems Consortium DHCP Client 4.4.1

Copyright 2004-2018 Internet Systems Consortium.

All rights reserved.

For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth\_cli/aa:fd:16:47:4a:cf

Sending on LPF/eth\_cli/aa:fd:16:47:4a:cf

Sending on Socket/fallback

Created duid "\000\001\000\001(\246i\245\252\375\026GJ\317".

DHCPDISCOVER on eth\_cli to 255.255.255.255 port 67 interval 6

DHCPOFFER of 192.168.1.142 from 192.168.1.1

DHCPREQUEST for 192.168.1.142 on eth\_cli to 255.255.255.255 port 67

DHCPACK of 192.168.1.142 from 192.168.1.1

Received variables:

Hostname: afcdeabc7bc3

IP address: 192.168.1.142

Network: 192.168.1.0

Netmask: 255.255.255.0

Router: 192.168.1.1

Domain Name: mycompany.local

DNS server: 192.168.1.1

bound to 192.168.1.142 -- renewal in 39980 seconds.

The debugging script that we are passing with the -sf parameter is so that instead of actually modifying the whole network settings in this machine, we can see what information was received from the server.

In the debugging output we see that the options set in the configuration file were correctly sent to the client.

If you take a look at the dnsmasq log, you will see these additional lines:

Aug 11 10:57:44 dnsmasq-dhcp[1217]: DHCPDISCOVER(eth\_srv) aa:fd:16:47:4a:cf

Aug 11 10:57:44 dnsmasq-dhcp[1217]: DHCPOFFER(eth\_srv) 192.168.1.142 aa:fd:16:47:4a:cf

Aug 11 10:57:44 dnsmasq-dhcp[1217]: DHCPREQUEST(eth\_srv) 192.168.1.142 aa:fd:16:47:4a:cf

Aug 11 10:57:44 dnsmasq-dhcp[1217]: DHCPACK(eth\_srv) 192.168.1.142 aa:fd:16:47:4a:cf afcdeabc7bc3

So, we see that the server saw the request and replied with an address.

We also see that it knows that the machine with the MAC address of eth\_cli is called linux-instance. The domain of the network is set to mycompany.local. This means that we can query the IP address of linux-instance.mycompany.local and it should give us the address that was received via DHCP:

dig linux-instance.mycompany.local @localhost

Copied!

content\_copy

What did dnsmasq say?

Aug 11 11:00:35 dnsmasq[1217]: query[A] linux-instance.mycompany.local from 127.0.0.1

Aug 11 11:00:35 dnsmasq[1217]: forwarded linux-instance.mycompany.local to 169.254.169.254

Aug 11 11:00:35 dnsmasq[1217]: reply linux-instance.mycompany.local is NXDOMAIN

When dnsmasq assigns the dynamic addresses it also keeps track of the names of the machines in the network. This is so if other machines request those names, dnsmasq will recognize the associated addresses.

**Changing the configuration**

Alright, so we've now seen what dnsmasq is doing and experimented obtaining an IP address from it. Our company's request to assign fixed IPs to the new server still needs to be fulfilled. This kind of assignment is done using the MAC address of the network card. We need to know the MAC addresses of the servers and which IP addresses to assign to them:

* aa:bb:cc:dd:ee:b2 - 192.168.1.2
* aa:bb:cc:dd:ee:c3 - 192.168.1.3
* aa:bb:cc:dd:ee:d4 - 192.168.1.4

Additionally, we know that we will be setting up more servers in the future, so we want to leave the first 20 IP addresses outside of the dynamic range.

We will use the dhcp-host configuration option to assign a specific address to a machine. The format of this option is to list the MAC address of the machine, followed by a comma, and then the IP address being assigned to it. For example, for the first server in the list, it would look like this:

dhcp-host=aa:bb:cc:dd:ee:b2,192.168.1.2

dhcp-host=aa:bb:cc:dd:ee:c3,192.168.1.3

dhcp-host=aa:bb:cc:dd:ee:d4,192.168.1.4

Copied!

content\_copy

To edit the configuration again, stop the server using the service command.

sudo service dnsmasq stop

Copied!

content\_copy

Then let's re-open the configuration file using *nano*.

sudo nano /etc/dnsmasq.d/mycompany.conf

Copied!

content\_copy

We need to change the dhcp-range option and add dhcp-host lines:

* Change the dhcp-range option so that it starts with 192.168.1.20 instead of 192.168.1.2
* Change the 24h lease time to 6h
* At the end of this file, add three dhcp-host lines, one for each of the servers that need to be configured, as defined above.

# This is the interface on which the DHCP server will be listening to.

interface=eth\_srv

# This tells this dnsmasq to only operate on that interface and not operate

# on any other interfaces, so that it doesn't interfere with other running

# dnsmasq processes.

bind-interfaces

# Domain name that will be sent to the DHCP clients

domain=mycompany.local

# Default gateway that will be sent to the DHCP clients

dhcp-option=option:router,192.168.1.1

# DNS servers to announce to the DHCP clients

dhcp-option=option:dns-server,192.168.1.1

# Dynamic range of IPs to use for DHCP and the lease time.

dhcp-range=192.168.1.20,192.168.1.254,6h

dhcp-host=aa:bb:cc:dd:ee:b2,192.168.1.2

dhcp-host=aa:bb:cc:dd:ee:c3,192.168.1.3

dhcp-host=aa:bb:cc:dd:ee:d4,192.168.1.4

log-queries

log-facility=/var/log/dnsmasq.log

Once you've done this, press "Ctrl-X" to exit the editor. It will ask you to save your changes. Press "Y" for yes and then enter at the filename prompt.

Click Check my progress to verify the objective.

Reduce Dynamic IP Range

Check my progress

Again, let's verify our configuration file.

sudo dnsmasq --test -C /etc/dnsmasq.d/mycompany.conf

Copied!

content\_copy

dnsmasq: syntax check OK.

If instead you get an error like: "dnsmasq: bad hex constant at line 21 of /etc/dnsmasq.d/mycompany.conf", it means that you have an error in the line indicated. Open the file with nano once again, and review and fix whatever is wrong in that line.

Click Check my progress to verify the objective.

New Servers have Fixed Addresses

Check my progress

Once we are sure that the configuration has the right syntax, we can start the service back up with the same command as before:

sudo service dnsmasq start

Copied!

content\_copy

Now, we want to verify that the lines we added to set specific IP addresses are in effect. Given that we are using a test instance with a simulated network environment, we can test this change by setting the MAC address of our virtual network interface to match one of the addresses specified in the config file.

**Important**: You shouldn't change the MAC address of actual machines on the network, as this could lead to many networking problems that are hard to debug. This technique can be used here because it's a testing simulation.

Let's change the MAC address of eth\_cli by using the ip link command.

sudo ip link set eth\_cli address aa:bb:cc:dd:ee:c3

Copied!

content\_copy

This command, as so many other Linux commands, doesn't generate any output if it succeeded. In order to see what it did, we need to look at the interface again, as we have done before.

sudo ip address show eth\_cli

Copied!

content\_copy

2: eth\_cli@eth\_srv: mtu 1500 qdisc noqueue state UP group default qlen 1000

link/ether aa:bb:cc:dd:ee:c3 brd ff:ff:ff:ff:ff:ff

We see that the MAC address (shown in the link/ether line) is now set to the one that corresponds to one of the new servers. Let's see if dnsmasq gives us the right IP address.

sudo dhclient -i eth\_cli -v -sf /root/debug\_dhcp.sh

Copied!

content\_copy

Internet Systems Consortium DHCP Client 4.4.1

Copyright 2004-2018 Internet Systems Consortium.

All rights reserved.

For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth\_cli/aa:bb:cc:dd:ee:c3

Sending on LPF/eth\_cli/aa:bb:cc:dd:ee:c3

Sending on Socket/fallback

DHCPREQUEST for 192.168.1.142 on eth\_cli to 255.255.255.255 port 67

DHCPREQUEST for 192.168.1.142 on eth\_cli to 255.255.255.255 port 67

DHCPDISCOVER on eth\_cli to 255.255.255.255 port 67 interval 4

DHCPOFFER of 192.168.1.3 from 192.168.1.1

DHCPREQUEST for 192.168.1.3 on eth\_cli to 255.255.255.255 port 67

DHCPACK of 192.168.1.3 from 192.168.1.1

Received variables:

Hostname: afcdeabc7bc3

IP address: 192.168.1.3

Network: 192.168.1.0

Netmask: 255.255.255.0

Router: 192.168.1.1

Domain Name: mycompany.local

DNS server: 192.168.1.1

bound to 192.168.1.3 -- renewal in 8244 seconds.

In this case, we see that as a first step the client tried to request the same IP address that it had before, (the DHCPREQUEST lines) but it got no reply. Eventually, it tried to get a new address, by doing a DHCPDISCOVER and for that it did get a reply, with the desired address.

What did dnsmasq say?

Aug 11 11:07:44 dnsmasq-dhcp[1435]: DHCPDISCOVER(eth\_srv) 192.168.1.142 aa:bb:cc:dd:ee:c3

Aug 11 11:07:44 dnsmasq-dhcp[1435]: DHCPOFFER(eth\_srv) 192.168.1.3 aa:bb:cc:dd:ee:c3

Aug 11 11:07:44 dnsmasq-dhcp[1435]: DHCPREQUEST(eth\_srv) 192.168.1.3 aa:bb:cc:dd:ee:c3

Aug 11 11:07:44 dnsmasq-dhcp[1435]: DHCPACK(eth\_srv) 192.168.1.3 aa:bb:cc:dd:ee:c3 afcdeabc7bc3

We've made the requested change to the DHCP configuration and verified that it works as expected.

If you were doing this in real life, now is when you would copy the configuration file to the production server.

**Conclusion**

Congrats! You've successfully experimented with DNS queries and modified the configuration of an existing DHCP server.

This was not a simple task and you've put in practice all your service management knowledge as well as your networking knowledge to be able to achieve this. Awesome!