Unknown Title

Legacy article:

Using_FOG_with_an_unmodifiable_DHCP_server/_Using_FOG_with_no_DHCP_server

Related articles:

Modifying existing DHCP server to work with FOG

BIOS and UEFI Co-Existence

dnsmasq's Roles in FOG

From the perspective of FOG, dnsmasq is used when there is an existing DHCP service on the network that must continue to be used and cannot be altered to support FOG. dnsmasq is a form of Proxy DHCP. It listens for DHCP requests (from hosts) and responses (from dhcp service). When a request and response is heard, dnsmasq "adds to" the response. For it's role in fog, it adds the next-server and file name options. These are known in Windows as DHCP Options 066 and 067.

Ideal scenarios for dnsmasq include:

- When you are unwilling or unable to turn off DHCP services on a consumer-grade piece of network equipment (such as an ISP-provided device or low-end store purchased device)
- When you are unwilling or unable to run DHCP on your FOG Server.
- When you do not have access or permission to change the DHCP service at your place of employment.
- When changes to your employer's DHCP service may be overly complex to perform.
- When errors made in the configuration of your employer's DHCP service could cause an unwanted, unplanned, or unexpected network outage.
- When you intend your FOG server to be portable.

How ProxyDHCP works

1. When a PXE client boots up, it sends a DHCP Discover broadcast on the network, which includes a list of information the client would like from the DHCP server, and some information identifying itself as a PXE capable device.

2. A regular DHCP server responds with a DHCP Offer, which contains possible values for network settings requested by the client. Usually a possible IP address, subnet mask, router (gateway) address, dns domain name, etc.
3. Because the client identified itself as a PXEClient, the proxyDHCP server also responds with a DHCP Offer with additional information, but not IP address info. It leaves the IP address assigning to the regular DHCP server. The proxyDHCP server provides the next-server-name and boot file name values, which is used by the client during the upcoming TFTP transaction.
4. The PXE Client responds to the DHCP Offer with a DHCP Request, where it officially requests the IP configuration information from the regular DHCP server.
5. The regular DHCP server responds back with an ACK (acknowledgement), letting the client know it can use the IP configuration information it requested.
6. The client now has its IP configuration information, TFTP Server name, and boot file name and it initiate a TFTP transaction to download the boot file.

Install dnsmasq on CentOS 7

Reference: https://forums.fogproject.org/topic/6376/install-dnsmasq-on-centos-7

Setting up DNSMasq on Centos 7 is pretty straight forward and can be done in about 10 minutes.

Use case(s):

1. You don't have administrative access to the dhcp server for your subnet/network (such as an ISP run router)
2. Your dhcp server is a basic one like what you might find if you use a home class internet router.

Here are the steps needed to setup dnsmasq on your FOG server running under Centos 7

1. Ensure that Centos is up to date

   yum upgrade -y

2. Install the service

   yum install dnsmasq -y

3. Create a config file for your FOG server

   vi /etc/dnsmasq.d/ltsp.conf (hint: I'm old school and use vi exclusively, you may use what ever editor you choose)

4. Paste in the following settings

```
# Don't function as a DNS server:
port=0
```

```
# Log lots of extra information about DHCP transactions.
log-dhcp

# Set the root directory for files available via FTP.
tftp-root=/tftpboot

# The boot filename, Server name, Server Ip Address
dhcp-boot=undionly.kpxe,,<fog_server_IP>

# Disable re-use of the DHCP servername and filename fields as extra
# option space. That's to avoid confusing some old or broken DHCP
clients.
dhcp-no-override

# inspect the vendor class string and match the text to set the tag
dhcp-vendorclass=BIOS,PXEClient:Arch:00000
dhcp-vendorclass=UEFI32,PXEClient:Arch:00006
dhcp-vendorclass=UEFI,PXEClient:Arch:00007
dhcp-vendorclass=UEFI64,PXEClient:Arch:00009

# Set the boot file name based on the matching tag from the vendor class
(above)
dhcp-boot=net:UEFI32,i386-efi/ipxe.efi,,<fog_server_IP>
dhcp-boot=net:UEFI,ipxe.efi,,<fog_server_IP>
dhcp-boot=net:UEFI64,ipxe.efi,,<fog_server_IP>

# PXE menu.  The first part is the text displayed to the user.  The
second is the timeout, in seconds.
pxe-prompt="Booting FOG Client", 1

# The known types are x86PC, PC98, IA64_EFI, Alpha, Arc_x86,
# Intel_Lean_Client, IA32_EFI, BC_EFI, Xscale_EFI and X86-64_EFI
# This option is first and will be the default if there is no input from
the user.
pxe-service=X86PC, "Boot to FOG", undionly.kpxe
pxe-service=X86-64_EFI, "Boot to FOG UEFI", ipxe.efi
pxe-service=BC_EFI, "Boot to FOG UEFI PXE-BC", ipxe.efi

dhcp-range=<fog_server_ip>,proxy
```

You must update the <fog_server_ip> values with the exact IP address of your FOG server.

5. Restart the dnsmasq service

   systemctl restart dnsmasq.service

6. Then ensure dnsmasq service starts on each boot.

    systemctl enable dnsmasq.service

For the copy and paste people like me here is the concise version.

```
yum upgrade -y
yum install dnsmasq -y
vi /etc/dnsmasq.d/ltsp.conf
<insert text>
<update settings in text>

systemctl restart  dnsmasq.service
systemctl enable dnsmasq.service
```

Advanced dnsmasq techniques

Reference: https://forums.fogproject.org/topic/8726/advanced-dnsmasq-techniques

Now lets say we have a computer that will not boot with the default ipxe.efi file, but instead we need the alternate intel.efi boot kernel. We'll add some dynamics to our above script so that for all computers except for our specific model ipxe.efi is sent to the client and when we pxe boot our specific client intel.efi is sent to just that computer.

I do have to post a caveat here. The uuid field "should" represent the device type for the model and not the unique and individual device (we could use the mac address for that). I have not tested like model computers to see if the uuid is an exact match. I do see references to that this field contains two parts the uuid and guid bits. We may need to parse those if I find that these numbers are not model specific.

In the script above we're going to add a new pattern match test just under the vendor class match. Modify the above script to look similar to this snippet.

```
# inspect the vendor class string and match the text to set the tag
dhcp-vendorclass=BIOS,PXEClient:Arch:00000
dhcp-vendorclass=UEFI32,PXEClient:Arch:00006
dhcp-vendorclass=UEFI,PXEClient:Arch:00007
dhcp-vendorclass=UEFI64,PXEClient:Arch:00009

#UUID for a Dell e6230 I tested (this info was gleaned from the dnsmasq log
file that recorded
# a pxe boot session of this target computer
dhcp-match=set:e6230,97,00:44:45:4c:4c:38:00:10:36:80:4e:c4:c0:4f:4a:58:31
```

What this dhcp-match command does is set the flag e6230 to TRUE if dhcp option 97 {uuid/guid client identifier} if the data matches "00:44:45:4c:4c:38:00:10:36:80:4e:c4:c0:4f:4a:58:31" now if we determine a sub section of this uuid field is sufficient to identifiy the client we could shorten this pattern match to let say "00:44:45:4c:4c:38:00:10:36" if this properly identifies the e6230 (I simply don't know as of now).

Now that we have the match command we need to do something with that match. That is where the next line comes in. We'll add another dncp-boot line. First I'll mention a dhcp-boot line that we are NOT going to use and why. This line is close to what we want in the file config file

```
dhcp-boot=tag:e6230,intel.efi,192.168.112.24 192.168.112.24
```

To decode this line there is a conditional test (if (tag:e6230 == true) then Send "intel.efi" from the following tftp server 192.168.112.24. So if our pattern matches above and set the tag e6230 true then send intel.efi.

The reason why we **don't want to use this one** is because it will match as long as the uuid is the same. This means that the intel.efi boot file name will be sent if the computer is in uefi mode as well as bios (legacy) mode. To correct this behavior we'll add another conditional test which creates an AND condition. What we want is to send the intel.efi file name if e6230 and UEFI flags are set. This dhcp-boot line would look like this:

```
dhcp-boot=tag:UEFI,tag:e6230, intel.efi, 192.168.112.24, 192.168.112.24
```

So this line will match when the UEFI tag is true (set by the vendor class match of "dhcp-vendorclass=UEFI,PXEClient:Arch:00007") and the e6230 tage is true.

Remember I said above the order of the dhcp-boot lines appear to be important. The last match will win so we want to place this new dhcp-boot line at the bottom of the list. Adding this line in will make our total ltsp config file look like this.

```
port=0

# Log lots of extra information about DHCP transactions.
log-dhcp

# Set the root directory for files available via FTP.
tftp-root=/tftpboot

# Disable re-use of the DHCP servername and filename fields as extra
# option space. That's to avoid confusing some old or broken DHCP clients.
dhcp-no-override

# inspect the vendor class string and match the text to set the tag
dhcp-vendorclass=BIOS,PXEClient:Arch:00000
dhcp-vendorclass=UEFI32,PXEClient:Arch:00006
dhcp-vendorclass=UEFI,PXEClient:Arch:00007
dhcp-vendorclass=UEFI64,PXEClient:Arch:00009

#UUID for a Dell e6230 I tested (this info was gleaned from the dnsmasq log
file that recorded
# a pxe boot session of this target computer
dhcp-match=set:e6230,97,00:44:45:4c:4c:38:00:10:36:80:4e:c4:c0:4f:4a:58:31
```

```
# Set the boot file name based on the matching tag from the vendor class
(above)
dhcp-boot=net:UEFI32,i386-efi/ipxe.efi,,192.168.112.24
dhcp-boot=net:UEFI,ipxe.efi,,192.168.112.24
dhcp-boot=net:UEFI64,ipxe.efi,,192.168.112.24

# Our test to ensure both the UEFI and e6230 tags are set.
dhcp-boot=tag:UEFI,tag:e6230, intel.efi, 192.168.112.24, 192.168.112.24

# The boot filename, Server name, Server Ip Address
dhcp-boot=undionly.kpxe,,192.168.112.24

# PXE menu.  The first part is the text displayed to the user.  The second is
the timeout, in seconds.
pxe-prompt="Booting FOG Client", 1

dhcp-range=192.168.112.24,proxy
```

Save the file and exit out of the editor. Then restart the dnsmasq service.

Its been a while since I posted this. This knowledge here and above has been gleaned from some google-fu searches and trial and error (the hacker's way) to come up with the above. I'm sure much of this thread is inaccurate and the rest is completely wrong. This information is content that I've been able to compile of the past 2 days of testing. If you discover any information is inaccurate in this thread, please DM me and I'll integrate it into this document.

An interesting fact I found while researching the dhcp-match command for dhcp option 97. For the dell computers the uuid string of '00:44:45:4c:4c:38:00:10:36:80:4e:c4:c0:4f:4a:58:31" If you discount the first 8 bits [00] (i.e just look at. 44:45:4c:4c) that spells dell in hex ascii.

Match filter troubles

This last post is about the troubles I had when trying to build the match filter. I knew from past experieences that there was a uuid field and that data was sent with the initial dhcp request. I first saw this information when you pxe boot a target computer in bios (legacy) mode. It is displayed on the screen with the dhcp server's address, target computer's IP, netmask and gateway information. But usually it flies off the screen so quick its hard to document since its so long.

I did a little research on this dhcp option (97 client-identifier) field and this is what I found in the RFQ that describes these dhcp fields. Here is a snippet of the rfq (note this is not my intellectual property only a reprint from the original RFC-4361 https://tools.ietf.org/html/rfc4361)

```
   DHCPv4 clients that support more than one network interface SHOULD
   use the same DUID on every interface.  DHCPv4 clients that support
```

```
   more than one network interface SHOULD use a different IAID on each
   interface.
```

I did have some trouble getting the pattern match just right (inserted correctly below).

```
dhcp-match=set:e6230,97,00:44:45:4c:4c:38:00:10:36:80:4e:c4:c0:4f:4a:58:31
```

My first attempt at the text to match came from the dhcp request in wireshark. This dhcp option 97 was presented as "4c:4c:45:44:00:38:36:10:80:4e:c4:c0:4f:4a:58:31" in wireshark. So I pasted that into the dhcp-match and the match failed so the action never fired. (!!). Looking now at the raw data wireshark presented the proper information, according to the RFC, was in the dhcp packet it was not just presented on the screen.

SO I knew the log-dhcp option was set in the dnsmasq file, I checked the /var/log/syslog file and there was all of the dhcp information I was searching for, except... the dhcp option 97 line conained "00:44:45:4c:4c:38:00:10:36:80:4e:c4:c0:4f:4a..." (!!) its incomplete!

So being the hacker I am I merged the information from wireshark with the information from the dnsmasq log to produce the final match filter.

```
#From wireshark
   4c:4c:45:44:00:38:36:10:80:4e:c4:c0:4f:4a:58:31
#From syslog
00:44:45:4c:4c:38:00:10:36:80:4e:c4:c0:4f:4a...

#Produced
00:44:45:4c:4c:38:00:10:36:80:4e:c4:c0:4f:4a:58:31
```

Looking at it now I'm not even sure why it worked. I know the UUID/IAD is constructed with two parts. And based on the number from wireshark I can see there is a big endian little endian thing going on for the UUID. but that doesn't explain how the IAD part is correct.

I'm not sure where the 00: prefix comes from the number too. I know the first 4 letters should spell dell for dell computers. If you watch the bios boot screen you can see the UUID number presented there is something like 44454c4c4544-0038-3610-804ec4c04f4a5831 (hint it goes very fast), but you can see lines up with what dnsmasq reported minus the leading 00. It would be interesting to know how dell decides on the UUID for a specific model. I'm sure there is some encoding going on.

Compiling dnsmasq 2.76 if you need uefi support

Reference: https://forums.fogproject.org/topic/8725/compiling-dnsmasq-2-76-if-you-need-uefi-support

There has been a brilliant bit of code added to dnsmasq 2.76 (May 2016) to provide / fix support for sending uefi boot information to uefi systems. As of now most up to date Linux Distros have this version of dnsmasq available for install.

In this tutorial I'll outline the steps required to compile and install this latest version of dnsmasq for common distributions of linux. I don't have access to every version and/or flavor so I'll only document what I've personally perform. I would encourage other, that can, document their experiences here with flavors/versions of linux that I don't cover.

Before you compile this updated version of dnsmasq be sure that you install the version of dnsmasq from your linux distributions, package repository. This way you will be sure that all of the supporting scripts and dependences have been installed. In the steps below we will just replace the dnsmasq binary with the latest compiled version.

Ubuntu 16.04 LTS based systems

Build system: Mint 18 x64 (Based on Ubuntu 16.04 LTS) (note the following instructions worked perfectly for Raspbian Jessie which is Debian based)

1. First we need to setup our build environment

   ```
   sudo apt-get update

   sudo apt-get install build-essential

   sudo apt-get install -y wget libdbus-1-dev libnetfilter-conntrack-dev idn
   libidn11-dev nettle-dev libval-dev dnssec-tools
   ```

2. Next we'll get the source code for dnsmasq 2.76

   ```
   wget http://www.thekelleys.org.uk/dnsmasq/dnsmasq-2.76.tar.gz
   ```

3. Extract the source code from the tar file

   tar -zxf dnsmasq-2.76.tar.gz

4. Change into the dnsmasq build directory

   cd dnsmasq-2.76

5. Lets update a few settings in the config file. I know there are other ways to go about this with command line switches, but I didn't

   sudo vi src/config.h

6. Find this section

   ```
   /* #define HAVE_LUASCRIPT */
   /* #define HAVE_DBUS */
   /* #define HAVE_IDN */
   /* #define HAVE_CONNTRACK */
   /* #define HAVE_DNSSEC */
   ```

7. Paste in these settings just below the above text

```
#define HAVE_DBUS
#define HAVE_IDN
#define HAVE_IDN_STATIC
#define HAVE_CONNTRACK
#define HAVE_DNSSEC
```

8. Save and exit the config.h file
9. We need to see where the current dnsmasq file is located. (NOTE: Please be sure that dnsmasq has already been installed in your linux distribution to ensure all of the dependences have been installed before we proceed)

   which dnsmasq

10. This command should respond with something like this:

```
# which dnsmasq
/usr/sbin/dnsmasq
```

11. The key bit of info here is that dnsmasq is installed in **/usr/sbin**. What we need to do is tell the install script to not place the dnsmasq files in the default location (according to dnsmasq of /usr/local/sbin) but to place the files where the distribution dnsmasq put them (/usr/sbin). SO in this case we want to overwrite the dnsmasq binary in /usr/sbin. To do this we need to update the prefix variable in the Makefile (compiler instruction file)
12. Since we know where dnsmasq is now, lets go and update the Makefile to reflect the location where we dnsmasq installed

    sudo vi Makefile

13. Search for this line and change

```
PREFIX        = /usr/local
# To this
PREFIX        = /usr
```

14. Save and exit out of the Makefile
15. Lets backup the original dnsmasq executable just in case...

    sudo cp /usr/sbin/dnsmasq /usr/sbin/dnsmasq.old

16. Ok here's where we create and install the latest version of dnsmasq

    sudo make install

    At this point the compiler will dig through the source code and compile the dnsmasq program. Hopefully it will compile and install without errors.

17. Once the install is done lets ensure that the right version of dnsmasq is found first in the search path.
18. Key in the following

    dnsmasq -v The output should look like this:

    ```
    Dnsmasq version 2.76  Copyright (c) 2000-2016 Simon Kelley
    Compile time options: IPv6 GNU-getopt DBus no-i18n IDN DHCP DHCPv6 no-Lua
    TFTP conntrack ipset auth DNSSEC loop-detect inotify

    This software comes with ABSOLUTELY NO WARRANTY.
    Dnsmasq is free software, and you are welcome to redistribute it
    under the terms of the GNU General Public License, version 2 or 3.
    ```

19. Ensure the version displays 2.76 if so you are all set
20. The last and final step is to ensure that the application runs when the service is called.

    sudo service dnsmasq restart

21. If the service starts correctly (no errors) then you're done.
22. If you question if dnsmasq is running the proper version you can always inspect /var/log/syslog for any dnsmasq error messages.