Chatter on the Wire:

A look at DHCPv6 traffic

by Eric Kollmann

v.0.1
November 2010

Notes

- This paper is somewhat different than the others I've written in the past as it is being done for a school project. So it will be done more factual and to the point in some areas. It may get expanded on in the future after it has been entered for a grade.
- This is the third paper in my “Chatter on the Wire” series. One of these days I'll put them all together in one big article!
- As far as I'm aware this is the only research being done on DHCPv6 fingerprinting. I'm sure that will change in the near future, but as of today, this is it.

Contents

Introduction

You might wonder why research needs done on DHCPv6, specifically on how to use it for passive fingerprinting as we already can do this with DHCPv4, TCP, and a ton of other protocols. While that is all true DHCPv4 is, at least in theory, on its way out the door. That probably won't happen for years now, but sooner or later all devices will be doing IPv6. When this happens DHCPv6 clients will be the new norm and some of our old passive fingerprinting methods will no longer be applicable.

IPv6 clients have a link-local address that allows them to communicate with each other on the same link only (same side of the router). This is what most Linux IPv6 clients that I've run into do by default. They self assign and unless configured to do so after the fact do not utilize DHCPv6 at all. This does help keep network chatter to a minimum, but may leave clients unable to talk to everywhere they want to go as they could be missing important pieces, such as the DNS servers that may reside on a different network!

For the sake of brevity, in this paper, we'll focus on client workstations that have a DHCPv6 client enabled and turned on by default for most of our examples herein. We'll stay away from the link-local addresses as I don't see a good way to utilize this in a fingerprinting approach at this time. Some of the OS's that implement and turn on a DHCPv6 client by default are most of the modern flavors of Windows and the latest non windows system I've found is OpenSolaris.

For testing purposes, in my lab, I did go in on some Linux systems and change it over to enable IPv6 and turn on the DHCPv6 client. This was a bit time consuming and hit or miss on how fast afterwards it would seem to take effect. Restarting the network service would eventually trigger the DHCPv6 client to send out SOLICIT messages though. Most systems tested were of the Live CD flavor where possible. This helped to reduce the amount of time required per system testing.

The main purpose of this research was to get me familiar enough with DHCPv6 that I felt comfortable writing a new module for Satori to parse and utilize the information gathered in a new fingerprinting module. While the fingerprints are still sparse I believe I've achieved my main goal and feel comfortable enough to start offering it out to those utilizing Satori for passive fingerprinting.

Throughout the rest of this paper we'll look at the RFCs, the flow of traffic and the packet structure in DHCPv6. We'll look at what has been useful so far for fingerprinting and what may be useful to cross check and possibly use in the future!

RFC History for DHCPv6

You may be wondering why we care about RFCs in a research paper on DHCPv6 fingerprinting. The RFCs lay the groundwork for what is supposed to happen when a client or server talk to each other. What we'll see as we move forward through this is that each client and server interpret these RFCs in their own way. With DHCPv6 I have not seen some of the major misinterpretations that I did in v4, but we still need to understand the format of the protocol before we can utilize it to fingerprint a device.

Like DHCPv4 and any protocol for that matter, the RFCs for DHCPv6 have changed a little over the years. The nice thing about DHCPv6 is there does not appear to have been as many updates and changes as there are in some of the other protocols. This could be because it is new enough that a ton of changes are not needed, or perhaps, which I like to believe, they have thought about many more of the issues ahead of time and tried to implement around these issues!

The main RFC we care about is RFC 3315 – "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)". It was written in July 2003. This gives the basics of the different options a client can send and the format of the packets along with the foundation of DHCPv6.

There are numerous other RFCs that add functionality and others that have been updated due to changes that are now happening to support Ipv6. Some of these RFCs are:
3319 – Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers
3633 – IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6
3646 – DNS Configuration options for Dynamic Host Configuration Protocol for Ipv6 (DHCPv6)
3736 – Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6
5007 – DHCPv6 Leasequery
4361 – Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)
5494 – IANA Allocation Guidelines for the Address Resolution Protocol (ARP)

These last 2 are actually RFCs that update 3315. You may wonder how a DHCPv4 RFC can update a DHCPv6 RFC? Well if you really want to know you'll need to dig into these RFCs yourself as that is a whole area where we don't need to go for DHCPv6 fingerprinting. But after going through RFCs again after a 3 year break I have more admiration for the people that write them and update them!

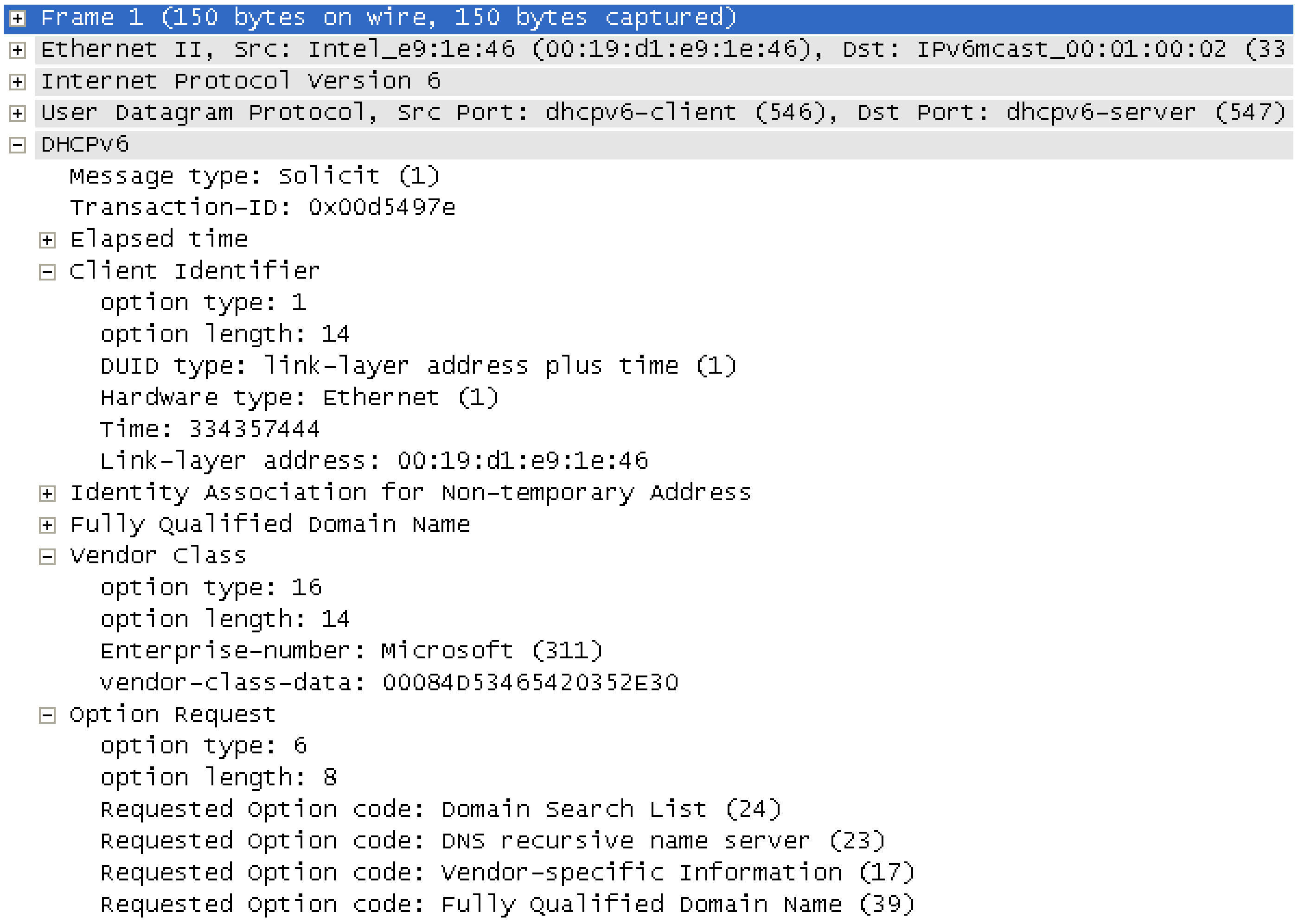
One other note on the aforementioned RFCs. 5007 would be a good one to look over for what implications or safeguards are put into place by the RFC and then again by the servers that implement it to protect unsavory characters from query a server to find out more about clients using DHCP on that network.

```
   One important motivating example is that the LEASEQUERY message
   allows access concentrators to query DHCP servers to obtain location
   information of broadband access network devices.
```

This is a very nice feature, as long as only those that should be able to do this can. Think if there is a new exploit against Linksys Routers and a 3rd party could query for this information. Basically the ability to ask for all Linksys routers and be handed back a list on a silver platter. Something to look deeper into in the future, but not required for our current project.

Format of a DHCPv6 Packet

Before we get any deeper into this lets look at a typical DHCPv6 packet. This one happens to be from a Windows 7 system.



```
Frame 1 (150 bytes on wire, 150 bytes captured)
Ethernet II, Src: Intel_e9:1e:46 (00:19:d1:e9:1e:46), Dst: IPv6mcast_00:01:00:02 (33
Internet Protocol Version 6
User Datagram Protocol, Src Port: dhcpv6-client (546), Dst Port: dhcpv6-server (547)
DHCPv6
  Message type: Solicit (1)
  Transaction-ID: 0x00d5497e
  Elapsed time
  Client Identifier
    option type: 1
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: Ethernet (1)
    Time: 334357444
    Link-layer address: 00:19:d1:e9:1e:46
  Identity Association for Non-temporary Address
  Fully Qualified Domain Name
  Vendor Class
    option type: 16
    option length: 14
    Enterprise-number: Microsoft (311)
    vendor-class-data: 00084D53465420352E30
  Option Request
    option type: 6
    option length: 8
    Requested Option code: Domain Search List (24)
    Requested Option code: DNS recursive name server (23)
    Requested Option code: Vendor-specific Information (17)
    Requested Option code: Fully Qualified Domain Name (39)
```

For those of you who have looked deeply at DHCPv4 and are comparing that to DHCPv6 you'll notice that a lot of the legacy things are now gone. We don't have things like the magic cookie, we no longer know these as bootp packets, etc.

The different expanded areas above are ones we will touch on here shortly. Each of these areas provide useful information when trying to fingerprint a system or in identifying the system. Most of this we will cover on as we go through the rest of the paper, but as a quick overview:

- Client Identifier – typically we like to utilize this to verify who the client is we are fingerprinting. If the DHCP packet has been routed through a relay the MAC address in the Ethernet Header will be incorrect.
- Vendor Class – some OS's will provide us with a hint to their OS here.
- Option Request – Each client has a unique list/order it requests information for here.

How DHCPv6 works

DHCPv6 works a lot like DHCPv4 did, especially from a fingerprinting perspective. The basic concept is to find and request an IPv6 address and then have the ability to ask for other pieces of information you may need.

I think now is a good time to look back at my previous paper:

> "Part of the reason DHCP Fingerprinting works so well is because DHCP is a broadcast packet. Even with all of those nice expensive, high end switches, replacing all those cheap hubs, when that DHCP Client decides to come up and request an IP address, they broadcast out for everyone to hear (at least on their local segment) that "Here I am; Here's my MAC; Here's what IP I had before; Here's what I want". From a passive OS identification perspective how can I complain about this lovely "feature"?"

The general concept here has not changed, while v6 clients don't do broadcasts, they still do multicast which is enough for other machines on the network to pick up on the packets. Clients can still loudly proclaim "Here I am" and "Here is what I want". The "here is what I want" is completely dependent on what services they are interested in and this the crux of how we can sort out a Windows 7 host vs an OpenSolaris host. Each OS wants different types of information from the DHCPv6 Server. Even if by some chance they happened to request the same pieces, the odds are they'll request them in a different order.

DHCPv6 vs DHCPv4 Message types

DHCPv6 introduced, or more precisely replaced the message types from DHCPv4. These new message types are much more readable on what they mean. Some of these we will use or care about, others due to the complexity of parsing the data from the wire have been ignored for now.

DHCPv6 Message Type	**DHCPv4 Message Type**
SOLICIT (1)	DHCPDISCOVER
ADVERTISE (2)	DHCPOFFER
REQUEST (3), RENEW (5), REBIND (6)	DHCPREQUEST
REPLY (7)	DHCPACK/DHCPNAK
RELEASE (8)	DHCPRELEASE
INFORMATION-REQUEST (11)	DHCPINFORM
DECLINE (9)	DHCPDECLINE
CONFIRM (4)	NA
RECONFIGURE (10)	DHCPFORCERENEW
RELAY-FORWARD (12), RELAY-REPLY(13)	NA

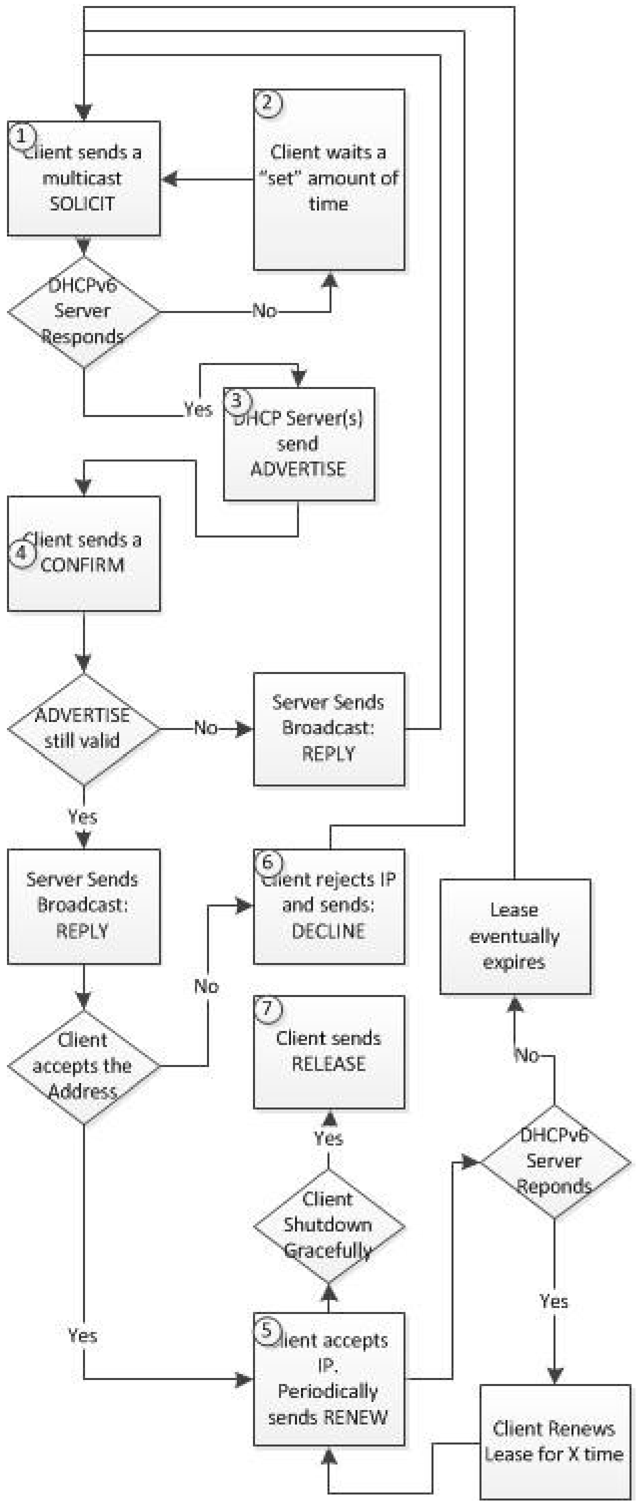

1 Client sends a multicast SOLICIT
2 Client waits a "set" amount of time
DHCPv6 Server Responds
No
Yes
3 DHCP Server(s) send ADVERTISE
4 Client sends a CONFIRM
ADVERTISE still valid
No
Server Sends Broadcast: REPLY
Yes
Server Sends Broadcast: REPLY
6 Client rejects IP and sends: DECLINE
Lease eventually expires
No
Client accepts the Address
7 Client sends RELEASE
No
Yes
DHCPv6 Server Reponds
Client Shutdown Gracefully
Yes
Yes
5 Client accepts IP. Periodically sends RENEW
Client Renews Lease for X time

Figure 1.1 – Flowchart of How DHCP packets flow

DHCPv6 traffic flow

The general flow of traffic has not changed much from DHCPv4 to DHCPv6. We have some changes such as multicast vs broadcast, but the basic process of requesting/renewing/etc is still basically the same.

At point 1, we can fingerprint the Client with the Options and if it has it in the Options Request Field. Information in each of these is typically unique or in a unique order. [More on how we do this to come shortly.]

At point 2, we can also fingerprint the Client. We won't go into this method in this paper, the same idea was presented in my previous paper and is under the "Hard way to fingerprint". Basically we do a timing test to see how often a client resends a SOLICIT message. While useful, there are much easier ways to do this already being covered!

At point 3, we can fingerprint the Server, this is typically only useful with SOHO type devices that do not allow you to change things. If you can change the Options that your Server advertises, any fingerprinting done here is near useless.

At point 4, we will use the same idea as point 1, except using CONFIRM packets instead of SOLICIT ones.

At point 5, we could try to verify the client sends its RENEW requests on time. While useful in a very small means, not worth researching at this time due to tracking purposes.

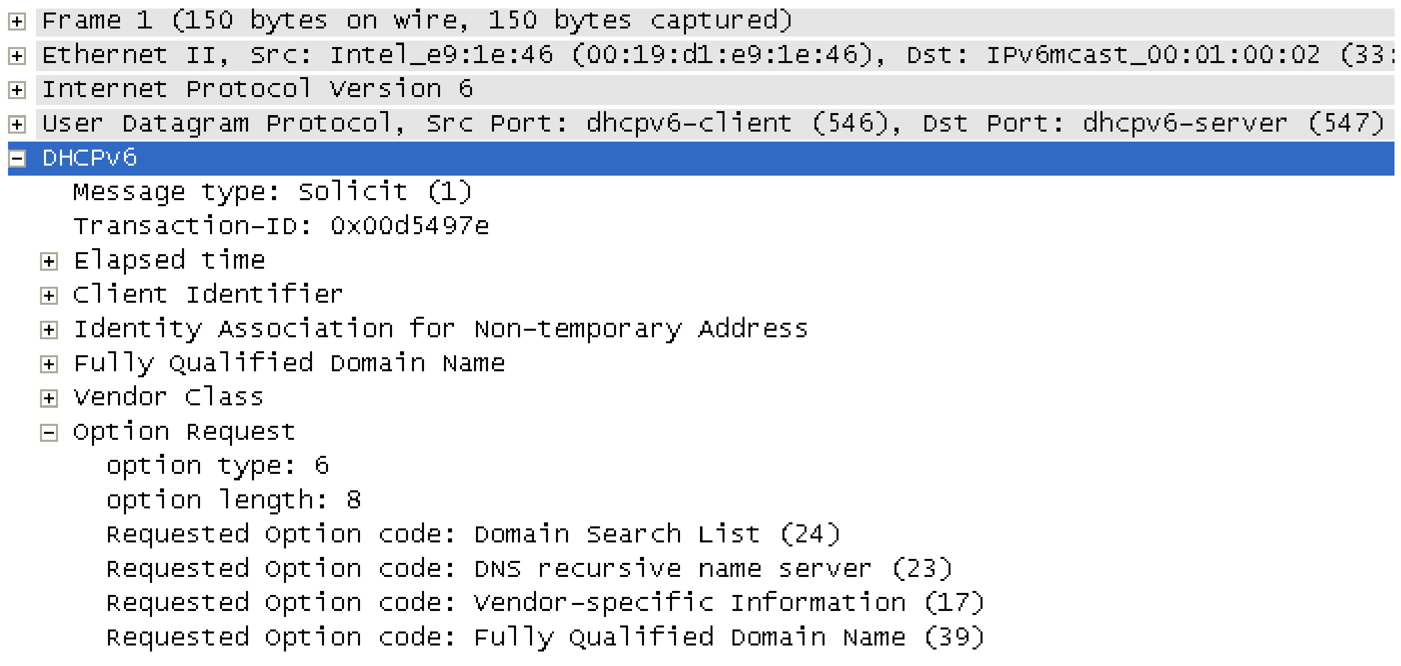
At point 6, we may find it to be useful, don't know yet. I have not seen any DECLINE packets to know if each client sends a unique set or if they all send the exact thing.

At point 7, we see something very few OSs behave nicely and do. That is sending a RELEASE packet. This is an advantage since you never know when you might spot a system. Also, add to that fact that most OS's don't do this, it brings down the list of systems we'll identify this way to just a few!

SOLICIT Packet

Let's first take a look at the initial boot up process of the machine. This will give us a general idea of what these packets look like that the DHCPv6 Clients and Servers are sending around to each other, look at how to read them, what the different sections are, and why we care about them.

Initially, upon boot up some clients will send a SOLICIT packet. Lets look at a typical one:



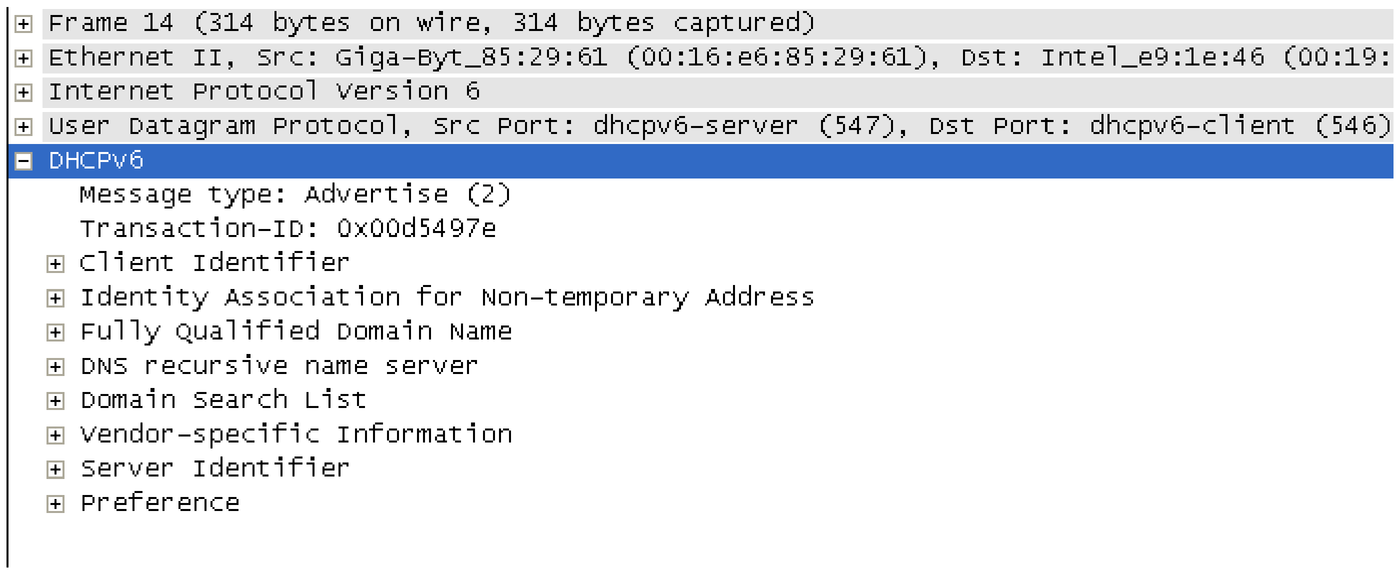
```
⊞ Frame 1 (150 bytes on wire, 150 bytes captured)
⊞ Ethernet II, Src: Intel_e9:1e:46 (00:19:d1:e9:1e:46), Dst: IPv6mcast_00:01:00:02 (33:
⊞ Internet Protocol Version 6
⊞ User Datagram Protocol, Src Port: dhcpv6-client (546), Dst Port: dhcpv6-server (547)
⊟ DHCPv6
    Message type: Solicit (1)
    Transaction-ID: 0x00d5497e
  ⊞ Elapsed time
  ⊞ Client Identifier
  ⊞ Identity Association for Non-temporary Address
  ⊞ Fully Qualified Domain Name
  ⊞ Vendor Class
  ⊟ Option Request
      option type: 6
      option length: 8
      Requested Option code: Domain Search List (24)
      Requested Option code: DNS recursive name server (23)
      Requested Option code: Vendor-specific Information (17)
      Requested Option code: Fully Qualified Domain Name (39)
```

Figure 1.2 - Typical SOLICIT Packet

Unlike DHCPv4 the client doesn't send this out as a broadcast packet, but now as an IPv6 multicast packet. You can also see that the ports for DHCPv6 have changed. We now use UDP 546 and 547 instead of UDP 67 and 68.

ADVERTISE Packet

Once a DHCPv6 Server receives a SOLICIT message it will send an ADVERTISE packet. Any DHCPv6 Server that hears the packet will send this ADVERTISE back to the client. It is up to the client to decide which one to request if it gets multiple advertisements. Most clients, in the v4 (DHCP and IP) world, took the first one they received. I have not seen anything about how machines deal with this in the v6 world, but I assume it is the same.



```
⊞ Frame 14 (314 bytes on wire, 314 bytes captured)
⊞ Ethernet II, Src: Giga-Byt_85:29:61 (00:16:e6:85:29:61), Dst: Intel_e9:1e:46 (00:19:
⊞ Internet Protocol Version 6
⊞ User Datagram Protocol, Src Port: dhcpv6-server (547), Dst Port: dhcpv6-client (546)
⊟ DHCPv6
    Message type: Advertise (2)
    Transaction-ID: 0x00d5497e
  ⊞ Client Identifier
  ⊞ Identity Association for Non-temporary Address
  ⊞ Fully Qualified Domain Name
  ⊞ DNS recursive name server
  ⊞ Domain Search List
  ⊞ Vendor-specific Information
  ⊞ Server Identifier
  ⊞ Preference
```

Figure 1.3 - Typical ADVERTISE packet sent from a DHCPv6 Server

Each DHCPv6 Server is going to do its own thing in regards to what Options it tells the Client about, but the main things are going to be IPv6 Address, Lease Time and DNS.

If you look at the SOLICIT and ADVERTISE packets you'll see that the options that each send to each other are different. In the SOLICIT message the client sent an Elapsed Time, Client Identifier, Identity Association, FQDN, Vendor Class, and an Option Request. Where on the ADVERTISE packet the server responded with Client Identifier, Identity Association, FQDN, DNS, Domain Search List, Vendor-specific Information, Server Identifier, Preferences. Each device is either requesting or sending what it thinks it or the other end needs! Ultimately a client can request whatever information it wants from the server throughout this process, but the server may not have that information or may not be configured to hand it out.

Also as a reminder, the Client may receive more than one ADVERTISE message. It just depends on your network. If this were to happen the Client must make a decision on which ADVERTISEment to accept. It may do this by which one answered first, or it may make that decision based on the Options it received from the Server. This is going to be very client specific.

Other DHCPv6 Message Types

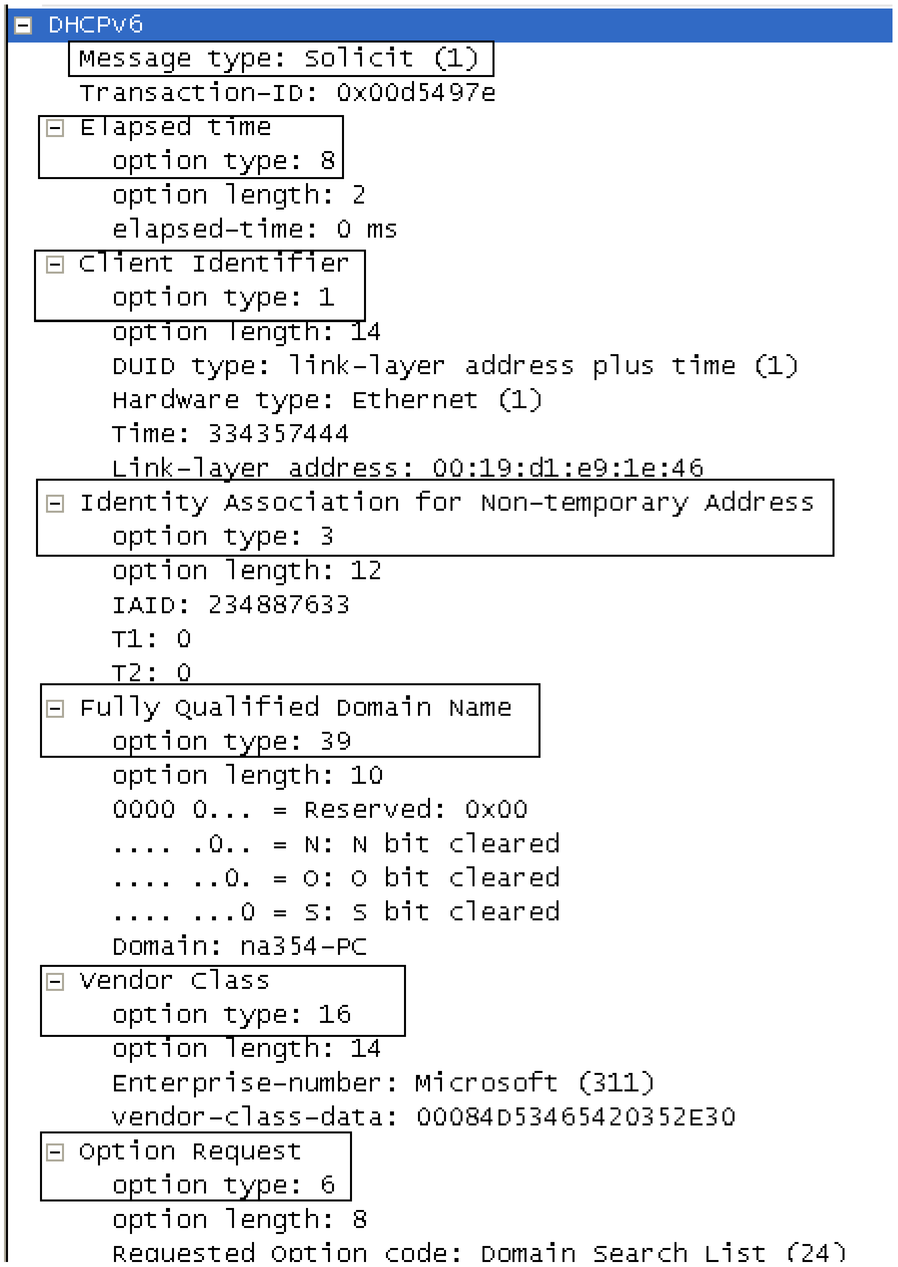
The REQUEST, REPLY, RELEASE and others can all also be used for fingerprinting purposes. The concept is the same as is the general format of the packets.

One major change in DHCPv6 that I've run into is that with RELAY-FORWARD and RELAY-REPLY messages (and possibly others) is that options can be buried inside of other options. Think of it like the russian dolls, one inside of the other. Parsing that for fingerprinting purposes became too cumbersome and is something I need to go back to, but since I have never run into this in my test scenario, only in a few wild packets, I've not spent extensive time trying to understand how it packs them all together though the RFC actually explains this.

The easy way to fingerprint

Using all Options

In the DHCPv4 fingerprinting world most other people doing fingerprinting have still not caught on to utilizing all the options as a list and to use them for fingerprinting. The advantage of this is we can typically group systems a bit better this way. We may not get an exact system with it, but we can get a family of systems, which depending on our other database(s) may be all we get! Lets once again look at a DHCPv6 packet below:



```
DHCPv6
  Message type: Solicit (1)
  Transaction-ID: 0x00d5497e
  Elapsed time
    option type: 8
    option length: 2
    elapsed-time: 0 ms
  Client Identifier
    option type: 1
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: Ethernet (1)
    Time: 334357444
    Link-layer address: 00:19:d1:e9:1e:46
  Identity Association for Non-temporary Address
    option type: 3
    option length: 12
    IAID: 234887633
    T1: 0
    T2: 0
  Fully Qualified Domain Name
    option type: 39
    option length: 10
    0000 0... = Reserved: 0x00
    .... .0.. = N: N bit cleared
    .... ..0. = O: O bit cleared
    .... ...0 = S: S bit cleared
    Domain: na354-PC
  Vendor Class
    option type: 16
    option length: 14
    Enterprise-number: Microsoft (311)
    vendor-class-data: 00084D53465420352E30
  Option Request
    option type: 6
    option length: 8
    Requested Option code: Domain Search List (24)
```

Each area that is boxed above is a separate option and each message type can be different (SOLICIT,

ADVERTISE, REQUEST, REPLY, etc). In this case our fingerprint would look something like:
Solicit, 8,1,3,39,16,6

This would be for a windows 7 system. As you'll be able to see in the fingerprint file for Satori, this is also the same fingerprint for Windows Server 2008, but not for Windows 2008 R2.

Note:
This was an interesting discovery. It was my understanding through other reading that Windows Vista and Windows 2008 shared the same underlying kernel and I assumed the same underlying TCP/IP stack. That research also indicated 2008 R2 and Windows 7 shared the same one. This research seems to show that the DHCPv6 stack between Windows 7 and 2008 is the same and that the 2008 R2 one is different. To further show this the DHCPv4 fingerprint for Windows 7 and 2008 R2 is the same and the fingerprint for 2008 and Vista is the same. So why the difference here? Don't know, but with this irregularity we can differentiate between Windows 7, 2008 and 2008 R2 where we couldn't before from the passive side!

Option Request

Option Request (option 6) replaced what we used in DHCPv4 to fingerprint (Option 55). Utilizing what I'll refer to as OptionRequest from now on out will give us the best information in regards to what OS is being run. This is an exact set of options that a client is requesting. One issue with it, is that as new software gets added to the system, it may request a new option here or there. With this change we may need to fall back to the Option List mentioned in the section before! In some cases OptionRequest will be the same across multiple OS's, or normally over multiple flavors of the same underlying kernel in the Linux world, but if the systems are truly a different OS, the collisions between them are normally few and far between. Utilizing the parameters requested in the OptionRequest, and the order in which they are requested will can typically identify the OS requesting the IP address.

We'll see more about this throughout the next few sections!

Windows 7, 2008, and 2008 R2

Notice we did not say Vista. While Vista will do IPv6, its DHCPv6 client does not, at least in my test environment send out any solicit packets.

During a normal startup process we'll see these packets from a Windows machine:

OS	Type	Options	Option Request
Window s 7	Solicit	8,1,3,39,16,6	24,23,17,39
Window s 7	Request	8,1,2,3,39,16,6	24,23,17,39
Window s 2008	Solicit	8,1,3,39,16,6	24,23,17,39
Window s 2008 R2	Solicit	8,1,3,16,6	24,23,17

With Windows 7 we get the same Option Request with both Solicit and with Request packets, but we get unique Options. This is a bit of a change from a lot of the systems we saw in the DHCPv4 world. Typically the Option Request equivalent (option 55 in DHCPv4) would have been the more unique fingerprint to base things off of, but in this case, with DHCPv6, we see that utilizing the Options as a whole is actually more useful!

OpenSolaris 2009.06

Next we'll look at Open Solaris:

OS	Type	Options	Option Request
Open Solaris 2009.06	Solicit	1,3,6,14,8	7,12,23,24,27,29

We mentioned how we check the Client Identifier field to get the client MAC in an earlier section. While not shown above, the client identification piece on OpenSolaris seems to fill in what others use for the MAC with all 0's. While this is not currently utilized in Satori, I could see some other product utilizing it in the future if there happened to be a collision between the above fingerprint and that of another OS. Basically using the fact that it is fingerprint X with client identification all 0's, then OS = OpenSolaris.

Fedora 13 and 14

Lastly we'll look at Fedora's latest 2 releases. Again the options requested are different than any we've seen so far, but lets compare them to each other and to their DHCPv4 counterparts for a bit more understanding.

OS	Type	DHCP	Options	Option 55
Fedora 13	Solicit	v6	1,6,8,3	23,24
Fedora 14	Solicit	v6	1,6,8,3	23,24
Fedora 13	Discover	v4	53,55	1,28,2,3,15,6,12,40,41,42,26
Fedora 13	Request	v4	53,54,55,55	1,28,2,3,15,6,12,40,41,42,26
Fedora 14	Request	v4	53,50,55	1,28,2,121,15,6,12,40,41,42,26,119,3

Notice how the number of options requested in v6 is so much smaller than that requested in v4. This is one thing that they found out over the years, the number of options out there was really too big and most were not needed! So while they've trimmed that down there is another difference worth looking at here. Both Fedora 13 and 14 fingerprint the same from a v6 perspective, but if they are still on a v4 network and sending requests out on it we can differentiate between them! We saw something similar to this with Windows before.

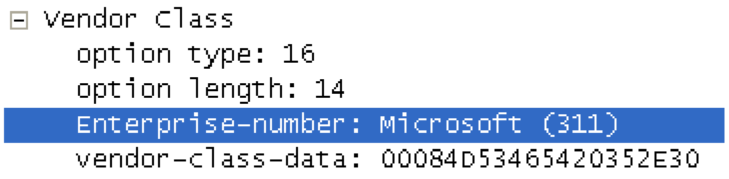
Beyond Option Requests – unique 'tells' to identification

Option 1

Option 1 - Client Identifier is used as a unique identifier to link a Client to its lease. Typically this will be the Clients MAC, but in some cases, as noted with OpenSolaris, the client may decide to put other information in here. Without about 500 different devices it is unknown how useful this parameter will pan out for other useful fingerprinting perspectives, but the fact that some systems seem to implement this differently than others it holds some promise as a secondary check site.

Option 16

Option 16 – Vendor Class is utilized for fingerprinting already, but there is now a way to double check and verify that the information presented in there is what it should be. In the picture below we see the Vendor Class. We have an Enterprise Number and the Vendor Class. If you convert that to ASCII you get MSFT 5.0. If someone was going to fake the vendor-class-data I assume they'd go to the trouble of putting in the correct Enterprise-number to go with it, but you never know.



```
⊟ Vendor Class
    option type: 16
    option length: 14
    Enterprise-number: Microsoft (311)
    vendor-class-data: 00084D53465420352E30
```

This Enterprise Number could be useful for some base fingerprinting. It would be simple enough to get a list of Enterprise Numbers though and build an initial list. That way,even if you didn't have any information in your fingerprint database about the options as a whole or the option request you could at least come back with a very generic guess at the system on the network. While this approach wouldn't give you a specific OS it gives you a starting point!

Other

There are bound to be quite a few other options out there that are currently under utilized or unique to a set or subset of systems. Perhaps Option XYZ is only ever asked for by OS QWR. Until the field starts to grow and others add to it we won't know.

Conclusion

The underlying concept of DHCPv6 fingerprinting is the same as v4. The 10-20 hours to write the new module test and fix it, along with the hours of testing new OS's while time consuming was well worth it. While I don't see IPv4, along with DHCPv4 going away any time in the near future it is good to get out ahead of this on the passive fingerprinting side. The database so far is by no means big, but now that the module is written and once the paper is published I assume this will change.

There are still many things that can be looked into deeper in the IPv6 world for fingerprinting along with DHCPv6 for that matter. But this is a good start and seems to be a solid way to go.

References

Kollmann, Eric – "Chatter on the Wire: A look at DHCP traffic" 2007.
http://myweb.cableone.net/~xnih/download/chatter-dhcp.pdf

RFC 3315 – "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)". http://tools.ietf.org/html/rfc3315