dhcpcanon Documentation

Release 0.8.3

juga

Apr 07, 2018

Contents

DHCP client disclosing less identifying information.

Python implementation of the DHCP Anonymity Profiles **RFC 7844** designed for users that wish to remain anonymous to the visited network minimizing disclosure of identifying information.

CHAPTER 1

Technologies

This implementation uses the Python Scapy Automata

1.1 What is the Anonymity Profile?

As the [**RFC 7844**] stats:

> Some DHCP options carry unique identifiers. These identifiers can enable device tracking even if the device administrator takes care of randomizing other potential identifications like link-layer addresses or IPv6 addresses. The anonymity profiles are designed for clients that wish to remain anonymous to the visited network. The profiles provide guidelines on the composition of DHCP or DHCPv6 messages, designed to minimize disclosure of identifying information.

1.2 What is DHCP?

From Wikipedia: The **Dynamic Host Configuration Protocol** (**DHCP**) is a standardized network protocol used on Internet Protocol (IP) networks. The DHCP is controlled by a DHCP server that dynamically distributes network configuration parameters, such as IP addresses, for interfaces and services. A router or a residential gateway can be enabled to act as a DHCP server. A DHCP server enables computers to request IP addresses and networking parameters automatically, reducing the need for a network administrator or a user to configure these settings manually. In the absence of a DHCP server, each computer or other device (eg., a printer) on the network needs to be statically (ie., manually) assigned to an IP address.

CHAPTER 2

Installation

See *Install dhcpcanon*

CHAPTER 3

Download

See *Download dhcpcanon*

CHAPTER 4

Bugs and features

If you wish to signal a bug or report a feature request, please fill-in an issue on the dhcpcanon issue tracker.

CHAPTER 5

Current status

Minimal version implemented, still to be improved.

See *TODO*

CHAPTER 6

Documentation for developers

Contributing to dhcpcanon

State of the Art

RFC7844 DHCPv4 restricted version summary, questions and dhcpcanon specification

Summary of questions regarding the RFCs and the implementations

Message types and options details in all layers

Installation and running cases

Minimising dhcpcanon privileges

dhcpcanon integration with network managers

dhcpcanon Python API Reference

dhcpcanon diagrams

Recommended documentation not included in this repository:

Related RFCs

RFC7844 comments and summary

Main Website

CHAPTER 7

License

CHAPTER 8

Acknowledgments

To all the persons that have given suggestions and comments about this implementation, the authors of the **RFC 7844**, the Prototype Fund Project of the Open Knowledge Foundation Germany and the Federal Ministry of Education and Research for funding partially this project.

CHAPTER 9

Contents:

9.1 Install dhcpcanon

The recommended way to install `dhcpcanon` is with your package source distribution, as it will also install other system files.

Currently is availabe for Debian unstable/testing. It can be installed with a package manager or in command line:

```
sudo apt install dhcpcanon
```

The main script will be installed in `/sbin/dhcpcanon`, a systemd service will be enabled and run by default, so there is no need to run anything manually.

Important: when running `dhcpcanon` the hardware address (MAC) should be randomized. You can use macchanger, macouflage or other.

9.2 Installation from source code in Debian/Ubuntu

In case you would like to have a newer version or it is not packaged for your distribution, you can install it from the source code.

Install system dependencies, in Debian/Ubuntu:

```
sudo apt install python3-dev
```

Obtain the source code:

```
git clone https://github.com/juga0/dhcpcanon/
```

Install `dhcpcanon` and system files:

```
sudo ./install.sh
```

9.2.1 for advanced users

Follow the two first steps in the previous paragraph.

To install `dhcpcanon` and the `systemd` service:

```
sudo make install WITH_SYSTEMD=true
```

In Debian this will install all the required files under `/usr/local`. `WITH_SYSTEMD` will install a systemd service and enable it, to run it:

```
systemctl start dhcpcanon
```

It's possible to also install support for udev:

```
sudo apt install sudo make install WITH_SYSTEMD=true
sudo make install WITH_SYSTEMD=true WITH_SYSTEMD_UDEV=true
```

And apparmor profile:

```
sudo apt install apparmor
sudo make install WITH_APPARMOR=true
```

In the case that you would like to install without root privileges, you can install it without the systemd service and you can specify an alternative location, for instance:

```
make --prefix=/home/user/.local install
```

Note however that without systemd `dhcpcanon` will need to be run with root privileges, while the systemd service drop `dhcpcanon` root privileges and only keeps the required network capabilities.

You would also need to install resolvconf-admin to be able to run it as non root user and set up DNS servers provided by the DHCP server. It will be possible to set up DNS servers with `systemd` too soon.

An alternative to do not run `dhcpcanon` with root privileges nor systemd, is to use ambient-rs wrapper and run:

```
RUST_BACKTRACE=1 ./target/debug/ambient \
-c NET_RAW,NET_ADMIN,NET_BIND_SERVICE \
/usr/bin/python3 -m dhcpcanon.dhcpcanon -v
```

9.3 Installation with pip

The pip package does not install either system files and it can be installed without root, but it still needs to be run as root, as commented in the last section.:

```
pip3 install dhcpcanon
```

In Debian this will install the files in `/home/youruser/.local` Note also that if you install it in a virtualenv, when executing `dhcpcanon` with `sudo`, won't use the virtualenv. To keep the virtualenv run it with:

```
sudo /pathtovirtualenv/bin/dhcpcanon
```

9.4 Installation for developers

It is recommended to install `dhcpcanon` in a python virtual environment.

Check https://virtualenv.pypa.io/en/latest/installation.html. In Debian:

```
sudo apt install python3-virtualenv
```

Create a virtual environment:

```
mkdir ~/.virtualenvs
virtualenv ~/.virtualenvs/dhcpcanonenv -p /usr/bin/python3
source ~/.virtualenvs/dhcpcanonenv/bin/activate
```

Get the sources:: git clone https://github.com/juga0/dhcpcanon

Install it:

```
pip3 install -e .
```

9.5 Download dhcpcanon

You can download this project in either zip or tar formats.

You can also clone the project with Git by running: git clone git://github.com/juga0/dhcpcanon

9.6 Running dhcpcanon

If `dhcpcanon` has be installed with systemd, it can be started with:

```
sudo systemctl start dhcpcanon
```

After installing, it can also be run manually:

```
sudo dhcpcanon
```

There is no need to pass any argument, most of the arguments are only used when `dhcpcanon` is called by other program (`systemd` or `gnome network manager`) and mimic the `dhclient` arguments.

You can specify which network interface to use passing it as an argument. Without specificying the network interface, it will use the active interface.

An useful argument when reporting bugs is `-v`.

An updated command line usage description can be obtained with:

```
dhcpcanon -h
```

9.7 Installation and running cases

9.7.1 system files

sbin/dhcpcanon-script systemd/dhcpcanon.service tmpfiles.d/dhcpcanon.conf systemd/network/90-dhcpcanon.link console_scripts -> /sbin/dhcpcanon

9.7.2 run cases

1. standalone without systemd, using -sp sbin/dhcpcanon-script
2. standalone without systemd, using resolvconf
3. standalone without systemd, using resolvconf-admin
4. launched with a wrapper, using -sp sbin/dhcpcanon-script
5. launched with a wrapper, sing resolvconf
6. launched with a wrapper, using resolvconf-admin
7. launched as systemd service, using systemd-resolved

9.7.3 install from

- setup.py: dhcpcanon-scriptresolvconf, resolvconf-admin and/or systemd need to be installed manually
- pip: dhcpcanon-scriptresolvconf, resolvconf-admin and/or systemd need to be installed manually
- Makefile
- Debian

9.8 TODO

[X] create debian package

[X] create documentation

[X] calculate retransmission times for DISCOVER

[X] create tests

[ ] (WIP) integrate with Network Manager

[ ] listen in several interfaces

[X] create systemd service

[X] create init.d daemon: Won't fix.

[X] limit privileges

[X] include MAC anonymization module: Debian package suggest it.

[X] create apparmor profile

[ ] implement IPv6

9.9 Contributing to dhcpcanon

We welcome contributions of any kind (ideas, code, tests, documentation, examples, . . .).

9.9.1 General contribution guidelines

- Any non-trivial change should contain tests.
- All the functions and methods should contain Sphinx docstrings which are used to generate the API documentation.

9.9.2 Code style guide

- We follow PEP8 Python Style Guide
- Use 4 spaces for a tab
- Use 79 characters in a line
- Make sure edited file doesn't contain any trailing whitespace
- You can verify that your modifications don't break any rules by running the `flake8` script - e.g. `flake8 dhcpcanon/edited_file.py` or `tox -e style`. Second command will run flake8 on all the files in the repository.

And most importantly, follow the existing style in the file you are editing and **be consistent**.

9.9.3 Docstring conventions

For documenting the API we we use Sphinx and reStructuredText syntax.

9.9.4 Contribution workflow

1. Open a new issue on our issue tracker

Go to our issue tracker and open a new issue for your changes there.

2. Fork our Github repository

Fork our Github git repository. Your fork will be used to hold your changes.

3. Create a new branch for your changes

For example:

```
git checkout -b <features/my_feature>
```

4. Make your changes

Commit often and rebase master

5. Write tests for your changes and make sure all the tests pass

Make sure that all the code you have added or modified has appropriate test coverage. Also make sure all the tests including the existing ones still pass using `tox`

```
tox
```

6. Open a Pull request

You can then push your feature branch to your remote and open a pull request.

9.9.5 Reporting a Vulnerability

Please do not report security issues using the public Issue tracker. Send a description of it to juga at riseup dot net. You are also encouraged to encrypt this email using GPG. The key can be found in the public servers.

This documentation is partly copied from libcloud contributing

9.10 State of the Art

on DHCP clients, network managers and libraries in Debian/Ubuntu

9.10.1 ISC-DHCP

Reference ISC implementation ISC License

homepage tar.gz

Debian DHCP clients

9.10.2 isc-dhcp-client

Debian default

debian debian source

9.10.3 network-manager built-in

9.10.4 systemd-networkd

`man 5 systemd.network` => DHCP options

9.10.5 udhcpc

Busybox implementation

debian

Debian network managers

9.10.6 Gnome Network Manager

Can use 3 DHCP clients: - ISC DHCP client: package *isc-dhpc-client*, binarry *dhclient* - systemd DHCP client - built-in DHCP client

debian

9.10.7 wicd

debian

Python DHCP libraries/tools

9.10.8 python-isc-dhcp-leases

Python module for reading dhcp leases files

debian

9.10.9 pydhcplib

Pure Python library.

GPL. Last updated XX. Commiters: 1.

pypi, repo, wiki debian

9.10.10 pydhcpd

DHCP command-line query and testing tool. Uses pydhcplib

GPL. Last updated: 2009

code

9.10.11 staticdhcpd

is an all-Python, RFC 2131-compliant DHCP server, with support for most common DHCP extensions and extensive site-specific customisation.

GPL. Last updated 12/03/2017. Commiters: +3

repo

9.10.12 dhquery

DHCP command line query and testing tool

code one github fork (updated 2016)

9.10.13 dhcpy6d

MAC address aware DHCPv6 server written in Python

Last updated 28/06/2017. Commiters: 2?

homepage repo doc debian

9.10.14 dhcpscapy

Simple DCHP client and server implemented with scapy

Last updated. 18/05/2014. Commiters: 1

repo

9.11 RFC7844 DHCPv4 restricted version summary, questions and `dhcpcanon` specification

This document is a more restrictive version summary of [**RFC 7844**], where the keywords (`key words` [**RFC 2119**]) commented in RFC7844 comments are actually replaced. Use `diff` to see specific differences between these two documents.

See *Summary of questions regarding the RFCs and the implementations* for a summary of the questions stated here.

Note:

- Extracts from the [**RFC 7844**] marked as literal blocks.
- Replacements are marked as parsed literal with the keyword replaced in bold

---

9.11.1 Message types

Note: See *Message types and options details in all layers* for a summary of the messages implementation

---

DHCP*

[**RFC 7844#section-3.1**]:

```
SHOULD randomize the ordering of options
```

```
If this can not be implemented
MUST order the options by option code number (lowest to highest).
```

DHCPDISCOVER

[**RFC 7844#section-3.**]:

```
MUST contain the Message Type option,
```

MUST NOT contain the Client Identifier option,

MUST NOT contain the Parameter Request List option.

MUST NOT contain any other option.

DHCPREQUEST

[RFC 7844#section-3.]:

```
MUST contain the Message Type option,
```

MUST NOT contain the Client Identifier option,

MUST NOT contain the Parameter Request List option.

MUST NOT contain any other option.

```
If in response to a DHCPOFFER,
MUST contain the corresponding Server Identifier option
MUST contain the Requested IP address option.

If the message is not in response to a DHCPOFFER (BOUND, RENEW),:
```

MUST NOT contain a Requested IP address option

DHCPDECLINE

[RFC 7844#section-3.]:

```
MUST contain the Message Type option,
MUST contain the Server Identifier option,
MUST contain the Requested IP address option;
```

MUST NOT contain the Client Identifier option.

- is it always broadcast?

DHCPRELEASE

[RFC 7844#section-3.]

To do not leak when the client leaves the network, this message type **MUST** NOT be implemented.

In this case, servers might run out of leases, but that is something that servers should fix decreasing the lease time.

DHCPINFORM

[RFC 7844#section-3.]:

```
MUST contain the Message Type option,
```

```
MUST NOT contain the Client Identifier option,
MUST NOT contain the Parameter Request List option.

It MUST NOT contain any other option.
```

9.11.2 Message Options

Client IP address (ciaddr)

[RFC 7844#section-3.2]:

```
MUST NOT include in the message a Client IP address that has been obtained
with a different link-layer address.
```

Requested IP Address Option (code 50)

[RFC 7844#section-3.3]

```
MUST NOT use the Requested IP address option in DHCPDISCOVER messages.
```

```
MUST use the option when mandated (DHCPREQUEST)

If in INIT-REBOOT:
```

```
MUST perform a complete four-way handshake, starting with a DHCPDISCOVER
```

- This is like not having INIT-REBOOT state?:

  ```
  If the client can ascertain that this is exactly the same network to which it was␣
  ↪previously connected, and if the link-layer address did not change,
  MAY issue a DHCPREQUEST to try to reclaim the current address.
  ```

- This is like INIT-REBOOT state?
- Is there a way to know `if` the link-layer address changed without leaking the link-layer?

Client Hardware Address Field

[RFC 7844#section-3.4]:

```
If the hardware address is reset to a new randomized value,
```

```
the DHCP client MUST use the new randomized value in the DHCP messages
```

The client should be restarted when the hardware address changes and use the current address instead of the permanent one.

Client Identifier Option (code 61)

[RFC 7844#section-3.5]

```
MUST NOT have this option
```

In the case that it would have this option because otherwise the server does not answer to the requests,:

```
DHCP
clients MUST use client identifiers based solely on the link-layer
address that will be used in the underlying connection.
```

Parameter Request List Option (PRL) (code 55)

[RFC 7844#section-3.6]

MUST NOT have this option

Host Name option (code 12)

[RFC 7844#section-3.7]

MUST NOT send the Host Name option.

Client FQDN Option (code 81)

[RFC 7844#section-3.8]

MUST NOT include the Client FQDN option

UUID/GUID-Based Client Machine Identifier Option (code 97)

[RFC 7844#section-3.9]:

```
Nodes visiting untrusted networks MUST NOT send or use the PXE options.
```

- And in the hypotetical case that nodes are visiting a "trusted" network, must this option be included for the PXE to work properly?

User and Vendor Class DHCP Options

[RFC 7844#section-3.10]

MUST NOT use the

```
Vendor-Specific Information option (code 43), the Vendor Class
Identifier option (code 60), the V-I Vendor Class option (code 124),
or the V-I Vendor-Specific Information option (code 125),
```

9.11.3 Operational considerations

[RFC 7844#section-5.]

```
Implementers SHOULD provide a way for clients to control when the
anonymity profiles are used and when standard behavior is preferred.
```

`dhcpcanon` does not currently implement the standard behavior described in [RFC 2131] in order to keep the implementation simple and because all existing implementations already implement it

9.11.4 Not specified in RFC7844, but in RFC2131

Probe the offered IP

[RFC 2131#section-2.2]:

```
the allocating
server SHOULD probe the reused address before allocating the address,
e.g., with an ICMP echo request, and the client SHOULD probe the
newly received address, e.g., with ARP.

 The client SHOULD perform a
check on the suggested address to ensure that the address is not
already in use.  For example, if the client is on a network that
supports ARP, the client may issue an ARP request for the suggested
request.  When broadcasting an ARP request for the suggested address,
the client must fill in its own hardware address as the sender's
hardware address, and 0 as the sender's IP address, to avoid
confusing ARP caches in other hosts on the same subnet.>>

The client SHOULD broadcast an ARP
reply to announce the client's new IP address and clear any outdated
ARP cache entries in hosts on the client's subnet.
```

- does any implementation issue an ARP request to probe the offered address?
- is it issued after DHCPOFFER and before DHCPREQUEST, or after DHCPACK and before passing to BOUND state?

Currently, there is not any probe

Retransmission delays

Sending DHCPDISCOVER [RFC 2131#section-4.4.1]:

```
The client SHOULD wait a random time between one and ten seconds to
   desynchronize the use of DHCP at startup.
```

- is the DISCOVER retranmitted in the same way as the REQUEST?

[RFC 2131#section-3.1]:

```
a client retransmitting as described in section 4.1 might retransmit the
DHCPREQUEST message four times, for a total delay of 60 seconds
```

[RFC 2131#section-4.4.5]:

```
In both RENEWING and REBINDING states,
if the client receives no response to its DHCPREQUEST
message, the client SHOULD wait one-half of the remaining
time until T2 (in RENEWING state) and one-half of the
remaining lease time (in REBINDING state), down to a
minimum of 60 seconds, before retransmitting the
DHCPREQUEST message.
```

[RFC 2131#section-4.1]:

```
For example, in a 10Mb/sec Ethernet
internetwork, the delay before the first retransmission SHOULD be 4
seconds randomized by the value of a uniform random number chosen
from the range -1 to +1

Clients with clocks that provide resolution
granularity of less than one second may choose a non-integer
randomization value.

The delay before the next retransmission SHOULD
be 8 seconds randomized by the value of a uniform number chosen from
the range -1 to +1.

The retransmission delay SHOULD be doubled with
subsequent retransmissions up to a maximum of 64 seconds.
```

- the delay for the next retransmission is calculated with respect to the type of DHCP message or for the total of DHCP messages sent indendent of the type?
- without this algorithm being mandatory, **it'd be possible to fingerprint the the implementation depending on the delay of the retransmission**
- how does other implementations do?

Selecting offer algorithm

[RFC 2131#section-4.2]:

```
DHCP clients are free to use any strategy in selecting a DHCP server
among those from which the client receives a DHCPOFFER message.

client may choose to collect several DHCPOFFER
messages and select the "best" offer.

If the client receives no acceptable offers, the client
may choose to try another DHCPDISCOVER message.
```

- what is a "no acceptable offer"?
- which are the "strategies" to select OFFER implemented?
- different algorithms to select an OFFER **could fingerprint the implementation**

[RFC 2131#section-4.4.1]:

```
The client collects DHCPOFFER messages over a period of time, selects
one DHCPOFFER message from the (possibly many) incoming DHCPOFFER
messages

The time
over which the client collects messages and the mechanism used to
select one DHCPOFFER are implementation dependent.
```

- Is it different the retransmission delays waiting for offer or ack/nak?, in all states?

Currently, the first OFFER is chosen

Timers

[RFC 2131#section-4.4.5]:

```
Times T1 and T2 are configurable by the server through options.  T1
defaults to (0.5 * duration_of_lease).  T2 defaults to (0.875 *
duration_of_lease).  Times T1 and T2 SHOULD be chosen with some
random "fuzz" around a fixed value, to avoid synchronization of
client reacquisition.
```

T1 is then calculated as:

```
renewing_time = lease_time * 0.5 - time_elapsed_after_request
range_fuzz = lease_time * 0.875 - renewing_time
renewing_time += random.uniform(-(range_fuzz), +(range_fuzz))
```

And T2:

```
rebinding_time = lease_time * 0.875 - time_elapsed_after_request
range_fuzz = lease_time - rebinding_time
rebinding_time += random.uniform(-(range_fuzz), +(range_fuzz))
```

The range_fuzz is calculated in the same way that `systemd` implementation does

- what's the fixed value for the fuzz and how is it calculated?
- The "fuzz" range is not specified, the fuzz chosen **could fingerprint** the implementation.

Leases

[RFC 7844#section-3.3]:

```
There are scenarios in which a client connecting to a network
remembers a previously allocated address, i.e., when it is in the
INIT-REBOOT state.  In that state, any client that is concerned with
privacy SHOULD perform a complete four-way handshake, starting with a
DHCPDISCOVER, to obtain a new address lease.  If the client can
ascertain that this is exactly the same network to which it was
previously connected, and if the link-layer address did not change,
the client MAY issue a DHCPREQUEST to try to reclaim the current
address.
```

- is there a way to know if the network the client is connected to is the same to which it was connected previously?

For the sake of simplicity and privacy `dhcpcanon` does not currently implement the INIT-REBOOT state nor reuse previously allocated addresses.

In future stages of `dhcpcanon` would be possible to reuse a previously allocated address. In order to do not leak identifying information when doing so, it would be needed:

- to keep a database with previously allocated addresses associated to:
  - the link network where the address was obtained (without revealing the MAC being used).
  - the MAC address that was used in that network

It is possible also that `dhcpcanon` will include a MAC randomization module in the same distribution package or would require it in order to start.

9.12 Summary of questions regarding the RFCs and the implementations

This is a summary of the questions stated in RFC7844 DHCPv4 restricted version summary

9.12.1 Message Options

Requested IP Address Option (code 50)

[**RFC 7844#section-3.3**]

- Is there a way to know `if` the link-layer address changed without leaking the link-layer?

9.12.2 Not specified in RFC7844, but in RFC2131

Probe the offered IP

[**RFC 2131#section-2.2**]

- does any implementation issue an ARP request to probe the offered address?
- is it issued after DHCPOFFER and before DHCPREQUEST, or after DHCPACK and before passing to BOUND state?

Retransmission delays

Sending DHCPDISCOVER [**RFC 2131#section-4.4.1**]

- is the DISCOVER retranmitted in the same way as the REQUEST

[**RFC 2131#section-3.1**], [**RFC 2131#section-4.4.5**], [**RFC 2131#section-4.1**]

- the delay for the next retransmission is calculated with respect to the type of DHCP message or for the total of DHCP messages sent indendent of the type?
- without this algorithm being mandatory, **it'd be possible to fingerprint the the implementation depending on the delay of the retransmission**
- how does other implementations do?

Selecting offer algorithm

[**RFC 2131#section-4.2**]

- what is a "no acceptable offer"?
- which are the "strategies" to select OFFER implemented?
- how many offers to wait for?
- different algorithms to select an OFFER **could fingerprint the implementation**

[**RFC 2131#section-4.4.1**]

- Is it different the retransmission delays waiting for offer or ack/nak?, in all states?

Timers

[RFC 2131#section-4.4.5]

- what's the fixed value for the fuzz and how is it calculated?
- The "fuzz" range is not specified, the fuzz chosen **could fingerprint** the implementation.

Leases

[RFC 7844#section-3.3]

- is there a way to know if the network the client is connected to is the same to which it was connected previously?

9.12.3 Not specified in any RFC

- is it needed to check that the ACK options match with the OFFER ones?
- is it needed to check that all options make sense?, which ones?

9.13 Message types and options details in all layers

9.13.1 DHCPDISCOVER

Always broadcast in AP:

```
Ehter: src=client_mac, dst="ff:ff:ff:ff:ff:ff"
IP: src="0.0.0.0", dst="255.255.255.255"
UDP: sport=68, dport=67
BOOTP: Client Hardware address (chaddr in scapy)
DHCP: Message Type option (message-type in scapy)
```

9.13.2 DHCPREQUEST

In SELECTING state: Broadcast in AP:

```
Ehter: src=client_mac, dst="ff:ff:ff:ff:ff:ff"
IP: src="0.0.0.0", dst="255.255.255.255"
UDP: sport=68, dport=67
BOOTP: Client Hardware address (chaddr in scapy)
DHCP: Message Type option (message-type in scapy)
DHCP: Server Identifier option (server_id in scapy, siaddr in server BOOTP offer)
DHCP: Requested IP option (requested_addr in scapy, yiaddr in server BOOTP offer)
```

In RENEWING state: Unicast to server id:

```
Ehter: src=client_mac, dst=server_mac
IP: src=client_ip, dst=server_ip
UDP: sport=68, dport=67
BOOTP: Client Hardware address (chaddr in scapy)
DHCP: Message Type option (message-type in scapy)
Client IP address (ciaddr=client_ip)?
```

In REBINDING state: broadcast:

```
Ehter: src=client_mac, dst="ff:ff:ff:ff:ff:ff"
IP: src="0.0.0.0", dst="255.255.255.255"
UDP: sport=68, dport=67
BOOTP: Client Hardware address (chaddr in scapy)
DHCP: Message Type option (message-type in scapy)
Client IP address (ciaddr=client_ip)?
```

9.13.3 DHCPDECLINE

Always broadcast?:

```
Ehter: src=client_mac, dst="ff:ff:ff:ff:ff:ff"
IP: src="0.0.0.0", dst="255.255.255.255"
UDP: sport=68, dport=67
BOOTP: Client Hardware address (chaddr in scapy)
DHCP: Message Type option (message-type in scapy)
DHCP: Server Identifier option (server_id in scapy, siaddr in server BOOTP offer)
DHCP: Requested IP option (requested_addr in scapy, yiaddr in server BOOTP offer)
```

9.13.4 DHCPRELEASE

Always unicast, is not being used:

```
Ehter: src=client_mac, dst=server_mac
IP: src=client_ip, dst=server_ip
UDP: sport=68, dport=67
BOOTP: Client Hardware address (chaddr in scapy)
DHCP: Message Type option (message-type in scapy)
DHCP: Server Identifier option (server_id in scapy, siaddr in server BOOTP offer)
```

9.13.5 DHCPINFORM

Always broadcast in Anonymity Profile, is not being used:

```
Ehter: src=client_mac, dst="ff:ff:ff:ff:ff:ff"
IP: src=client_ip, dst="255.255.255.255"
UDP: sport=68, dport=67
BOOTP: Client Hardware address (chaddr in scapy)
BOOTP: Client IP address (ciaddr=client_ip)
DHCP: Message Type option (message-type in scapy)
```

9.14 Minimising `dhcpcanon` privileges

Reasons why a DHCP client needs to run with root privileges:

- open sockets in privilege ports (68)
- open RAW sockets: to receive packets without having an IP set yet

- to set the IP offered

> **Note:** `dhcpcanon` does not need privileges to set up the IP, as that is done by a separated script, as `dhclient` does.

Possible solutions to minimise privileges and their associated problems:

1. drop privileges after BOUND DHCP state (sockets binded):

- problem: if the client stays connected until the renewing/rebinding time, privileges would be needed again and dropping privileges *temporally* it is not recommended [].
- possible solutions: do not implement RENEWING/REBINDING states.
  - problem: this would not be compliant with RFC 2131 nor 7844.
  - pro: in "usual" networks, if the client stays enough time connected to the network, the lease would expire it could just restart in the INIT state.

> **Todo:** which would be the associated problems to this solution?

2. wrapper with privileges to set linux network capabilities to the client, open sockets, then call the client inheriting the sockets:
   - problem: same as 1.

> **Note:** it's not possible to set net capabilities directly to a python script, they would need to be set to the python binary, but that would give the capabilities to any python script. Python binary could also be copied, set the capabilies, and that script call the client, but would have the same problem as giving the capabilities to the original python binary

3. `dhcpcanon` could call a binary with privileges to create the sockets every time it needs to do so. It's needed to change several parts of the current implementation.
4. to have the process be granted just the capabilities it needs, by the system-level process manager.

   This is already implemented with `systemd`
5. wrapper that does the same as in 4. without a system-level process manager. See section "wrapper to inherit capabilities"

   It could be solved with *infinity0's wrapper <https://github.com/infinity0/ambient-rs>* running:

```
RUST_BACKTRACE=1 ./target/debug/ambient -c NET_RAW,NET_ADMIN,NET_BIND_SERVICE /
↪usr/bin/python3 -m dhcpcanon.dhcpcanon -v
```

6. wrapper with privileges to disable linux Remote Path (RP) filter, open sockets, then call the client:

- problems:
  - it still needs root to change the default RP settings
  - it would only allow that the DHCP offers are received from other interfaces [], but still RAW sockets are needed to receive packets in the same interface that does not have an IP address yet
  - same as 1.

9.14.1 Wrapper to inherit capabilities

With `capsh`, `dhcpcanon` could be launched as another user and inherit only the required capabilities, in a similar way as `systemd.service` does:

```
capsh --caps=cap_net_raw,cap_net_bind_service,cap_net_admin+epi --keep=1 -- -c "mkdir␣
↪-p /run/dhcpcanon && cd /run/dhcpcanon && su -c 'exec /sbin/dhcpcanon enp0s25' -s /
↪bin/sh dhcpcanon"
```

`-s` is needed cause dhcpcanon shell is `/bin/false`

However this does not have capabilities to create the socket.

To show the capabilities that are actually inherited:

```
capsh --keep=1 --secbits=0x1C --caps=cap_net_raw,cap_net_bind_service,cap_net_
↪admin+epi  -- -c "mkdir -p /run/dhcpcanon && cd /run/dhcpcanon && su -c '/sbin/
↪capsh --print' -s /bin/sh dhcpcanon"
```

In `man capsh` `--securebits` is not documented, `securebits.h` has some documentation, but it seems to be needed a newer version of `libcap` as commented in this post

9.15 `dhcpcanon` integration with network managers

9.15.1 Integration with Gnome `Network Manager`

Gnome Network Manager has several components.

In Debian the service `NetworkManager` by default calls dhclient which in turn calls `nm-dhcp-helper`. Depending on the configuration, dhclient is called with the parameters:

```
/sbin/dhclient -d -q
-sf /usr/lib/NetworkManager/nm-dhcp-helper
-pf /var/run/dhclient-<interface>.pid
-lf /var/lib/NetworkManager/dhclient-<?>-<interface>.lease
-cf /var/lib/NetworkManager/dhclient-<interface>.conf
<interface>
```

Dclient calls `nm-dhcp-helper` via the `-sf` parameter, which seems to communicate back with `NetworkManager` via D-Bus.

`NetworkManager` can be configured to use dhcpcd or `internal`, as DHCP clients instead of `dhclient`.

```
FIXME: Configuring NetworkManager to use internal did not work
(why?). Is it using systemd DHCP client code? (libsystemd-network <https:/
↪/github.com/NetworkManager/NetworkManager/tree/master/src/systemd/src/
↪libsystemd-network`>`_
is included in ``NetworkManager source code, which is in systemd
code).

It does not work either with dhcpcd:
NetworkManager[12712]: <warn>  [1493146345.7994] dhcp-init: DHCP client␣
↪'dhcpcd' not available
```

Environment variables that `dhclient` returns

When `dhclient` call the script, by default `/sbin/dhcpcanon-script`, or when called by `NetworkManager`, `nm-dhcp-helper`, it pass environment variables.

```
FIXME: Are these variables documented somewhere?.
```

In `man dhclient-script` there is the list of values that the variable `reason` can take:

```
The following reasons
   are currently defined: MEDIUM, PREINIT, BOUND, RENEW, REBIND,  REBOOT,
   EXPIRE, FAIL, STOP, RELEASE, NBI and TIMEOUT.
```

But there are more variables. By setting `RUN=yes` in `/etc/dhcp/debug`, these variables are found in `/tmp/dhclient-script.debug`:

```
reason='PREINIT'
interface=
--------------------------
reason='REBOOT'
interface=
new_ip_address=
new_network_number=
new_subnet_mask=
new_broadcast_address=
new_routers=
new_domain_name=
new_domain_name_servers=
```

Looking at the code dhclient v4.3.5 there seem to be more variables.

Environment variables that `nm-dhcp-helper` gets

TBD

??

`dhcpcanon` required modifications

If `dhcpcanon` accepts the same arguments as `dhclient` and calls the script `nm-dhcp-helper` with the same environment variables as `dhclient`, it should be integrated.

```
FIXME: however for some reason this generates D-Bus errors.
```

`dhcpcanon` could also implement the D-Bus input/output that `NetworkManager` needs.

There's a NetworkManager D-Bus API specification.

There's also a Python API, python-networkmanager, so `dhcpcanon` could communicate directly with `NetworkManager` instead communicating with `nm-dhcp-helper`.

9.15.2 nm notes

Debugging:

[logging] level=DEBUG

It is not possible to set `dhcp-send-hostname` (Bug 768076 - No way to set dhcp-send-hostname globally) globally.

To modify `dhcp-send-hostname` per interface:

> nmcli connection modify "Wired connection" ipv4.dhcp-send-hostname no nmcli connection show "Wired connection"

Or the files: /etc/NetworkManager/system-connections/Wiredconnection

There is currently no way that when a new device is create it defaults to a configuration.

9.15.3 Integration with `wicd`

TBD

wicd

wicd documentation

9.16 dhcpcanon Python API Reference

`dhcpcanon.dhcpcapfsm`	
`dhcpcanon.dhcpcap`	
`dhcpcanon.dhcpcaplease`	
`dhcpcanon.clientscript`	
`dhcpcanon.timers`	Timers for the DHCP client implementation of the Anonymity Profile ([**RFC 7844**]).
`dhcpcanon.dhcpcaputils`	
`dhcpcanon.constants`	Constants for the DHCP client implementation of the Anonymity Profile ([**RFC 7844**]).
`dhcpcanon.conflog`	Logging configuration.

9.16.1 dhcpcapfsm module

9.16.2 dhcpcap module

9.16.3 dhcpcaplease module

9.16.4 clientscript module

9.16.5 timers module

Timers for the DHCP client implementation of the Anonymity Profile ([**RFC 7844**]).

`dhcpcanon.timers.`**`future_dt_str`**(*dt*, *td*)

.

`dhcpcanon.timers.`**`gen_delay_selecting`**()

Generate the delay in seconds in which the DISCOVER will be sent.

[**RFC 2131#section-4.4.1**]:

```
The client SHOULD wait a random time between one and ten seconds to
desynchronize the use of DHCP at startup.
```

dhcpcanon.timers.**gen_rebinding_time**(*lease_time*, *elapsed=0*)

.

dhcpcanon.timers.**gen_renewing_time**(*lease_time*, *elapsed=0*)

Generate RENEWING time.

[**RFC 2131#section-4.4.5**]:

```
T1
defaults to (0.5 * duration_of_lease).  T2 defaults to (0.875 *
duration_of_lease).  Times T1 and T2 SHOULD be chosen with some
random "fuzz" around a fixed value, to avoid synchronization of
client reacquisition.
```

dhcpcanon.timers.**gen_timeout_request_rebind**(*lease*)

.

dhcpcanon.timers.**gen_timeout_request_renew**(*lease*)

Generate time in seconds to retransmit DHCPREQUEST.

[**RFC 2131#section-4..4.5**]:

```
In both RENEWING and REBINDING states,
if the client receives no response to its DHCPREQUEST
message, the client SHOULD wait one-half of the remaining
time until T2 (in RENEWING state) and one-half of the
remaining lease time (in REBINDING state), down to a
minimum of 60 seconds, before retransmitting the
DHCPREQUEST message.
```

dhcpcanon.timers.**gen_timeout_resend**(*attempts*)

Generate the time in seconds in which DHCPDISCOVER wil be retransmited.

[**RFC 2131#section-3.1**]:

```
might retransmit the
DHCPREQUEST message four times, for a total delay of 60 seconds
```

[**RFC 2131#section-4.1**]:

```
For example, in a 10Mb/sec Ethernet
internetwork, the delay before the first retransmission SHOULD be 4
seconds randomized by the value of a uniform random number chosen
from the range -1 to +1.  Clients with clocks that provide resolution
granularity of less than one second may choose a non-integer
randomization value.  The delay before the next retransmission SHOULD
be 8 seconds randomized by the value of a uniform number chosen from
the range -1 to +1.  The retransmission delay SHOULD be doubled with
subsequent retransmissions up to a maximum of 64 seconds.
```

dhcpcanon.timers.**nowutc**()

.

9.16.6 dhcpcaputils module

9.16.7 constants module

Constants for the DHCP client implementation of the Anonymity Profile ([RFC 7844]).

dhcpcanon.constants.**PRL = b'\x01\x03\x06\x0f\x1f!+,./y\xf9\xfc'**

SD_DHCP_OPTION_SUBNET_MASK = 1 SD_DHCP_OPTION_ROUTER = 3 SD_DHCP_OPTION_DOMAIN_NAME_SERVER = 6 SD_DHCP_OPTION_DOMAIN_NAME = 15 SD_DHCP_OPTION_ROUTER_DISCOVER = 31 SD_DHCP_OPTION_STATIC_ROUTE = 33 SD_DHCP_OPTION_VENDOR_SPECIFIC = 43 SD_DHCP_OPTION_NETBIOS_NAMESERVER = 44 SD_DHCP_OPTION_NETBIOS_NODETYPE = 46 SD_DHCP_OPTION_NETBIOS_SCOPE = 47 SD_DHCP_OPTION_CLASSLESS_STATIC_ROUTE = 121 SD_DHCP_OPTION_PRIVATE_CLASSLESS_STATIC_ROUTE = 249 SD_DHCP_OPTION_PRIVATE_PROXY_AUTODISCOVERY = 252

9.16.8 conflog module

Logging configuration.

9.17 dhcpcanon diagrams

9.17.1 Finite State Machine diagram

9.17.2 Classes diagram

9.17.3 Packages diagram

9.17.4 Calls diagram

9.17.5 Organigram

This organigram does not reflect the current status of dhcpcanon, but as it should be changed

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index