**SOURCE CODE AND OUTPUT PICTURES**

**UNBOUND (Recursive Resolver)**

**Unbound.con{**

# Unbound configuration file for Debian. #Seethe unbound.conf(5) man page.

# See /usr/share/doc/unbound/examples/unbound.conf for a commented reference config file. # The following line includes additional configuration files from the

# /etc/unbound/unbound.conf.d directory. include: "/etc/unbound/unbound.conf.d/\*.conf'

**qname-minimisation.conf**

server:

# Send minimum amount of information to upstream servers to enhance #privacy.Only sends minimum required labels of the QNAME and sets # QTYPE to NS when possible.

# See RFC 7816 "DNS Query Name Minimisation to Improve Privacy" for # details. qname-minimisation: yes

**root-auto-trust-anchor-file.conf**

server:

# The following line will configure unbound to perform cryptographic # DNSSEC validation using the root trust anchor.

auto-trust-anchor-file: "/var/lib/unbound/root.key"

**Recursive.coof**

server:

# If no logfile is specified, syslog is used # Logging

logfile: "/var/log/unbound/unbound.log" verbosity: 5

use-syslog: no

log-time-ascii: yes log-queries: yes log-replies: yes

log-tag-queryreply: yes log-servfail: yes

# Interface Info

interface: 127.0.0.1 # Server listening at 127.0.0.1

port: 53

do-ip4: yes do-udp: yes do-tcp: yes

# Server listening on Port 53

# Send Responses in UDP # Send Responses in TCP

# May be set to yes if you have 1Pv6 connectivity

do-ip6: no# Since no 1Pv6 connectivity, currently set as no

# Leave this to no unless you have native 1Pv6. With 6to4 and

# Teredo tunnels your web browser should favor IPv4 for the same reasons prefer-ip6: no

# Use this only when you downloaded the list of primary root servers

# If you use the default dns-root-data package, unbound will find it automatically #root-hints: "/var/lib/unbound/root.hints"

# Trust glue only if it is within the server's authority harden-glue: yes

# Require DNSSEC data for trust-anchored zones, if such data is absent, the zone becomes #BOGUS

harden-dnssec-stripped: yes

# Harden against algorithm downgrade when multiple algorithms are advertised in the DS # record.

harden-alga-downgrade: yes

# Don't use Capitalization randomization as it known to cause DNSSEC issues sometimes use-caps-for-id: no

# Reduce EDNS reassembly buffer size.

# Suggested by the unbound man page to reduce fragmentation reassembly problems edns-buffer-size: 1472

# Rotates RRSet order in response (the pseudo-random number is taken from Ensure privacy of local IP ranges the query ID, for speed and thread safety).

# private-address: 192.168.0.0/16 rrset-roundrobin: yes

# Time to live minimum for RRsets and messages in the cache. If the minimum # kicks in, the data is cached for longer than the domain owner intended,

# and thus less queries are made to look up the data. Zero makes sure the # data in the cache is as the domain owner intended, higher values,

# especially more than an hour or so, can lead to trouble as the data in # the cache does not match up with the actual data anymore

cache-min-ttl: 600

#cache-max-ttl: 86400

# Have unbound attempt to serve old responses from cache with a TTL of 0 in # the response without waiting for the actual resolution to finish. The

# actual resolution answer ends up in the cache later on. serve-expired: yes

# Limit serving of expired responses to configured seconds after expiration. 0 disables the limit.

# This option only applies when serve-expired is enabled. A suggested value per RFC 8767 is between 86400 (1 day) and 259200 (3 days). The default is 0.

serve-expired-ttl: 86400 # one day, in second

# Fetch the DNSKEYs earlier in the validation process, when a DS record is

# encountered. This lowers the latency of requests at the expense of little more CPU usage. prefetch-key: yes

# Hides the id.server and hostname.bind queries, version.server and version.bind queries. hide-identity: yes

hide-version: yes

# Perform prefetching of close to expired message cache entries # This only applies to domains that have been frequently queried prefetch: yes

# One thread should be sufficient, can be increased on beefy machines. In reality for most users running on small networks or on a single machine

# it should be unnecessary to seek performance enhancement by increasing num-threads above 1.

num-threads: 1

# Ensure kernel buffer is large enough to not lose messages in traffic spikes so-rcvbuf: 2m

# Ensure privacy of local IP ranges private-address: 192.168.0.0/16 private-address: 169.254.0.0/16 private-address: 172.16.0.0/12 private-address: 10.0.0.0/8

private-address: fd00::/8 private-address: fe80::/10

# Remote control of unbound remote-control:

control-enable: yes

# unbound server key file.

server-key-file: "/etc/unbound/unbound\_server.key"

# unbound server certificate file.

server-cert-file: "/etc/unbound/unbound\_server.pem"

# unbound-control key file.

control-key-file: "/etc/unbound/unbound\_control.key"

# unbound-control certificate file.

control-cert-file: "/etc/unbound/unbound\_control.pem"

**PYTHON CODE**

import logging import argparse import dns.message import dns.name import dns.query import dns.rdata import dns.rdataclass import dns.rdatatype

from dns.exception import DNSException, Timeout # Root Servers IP addresses as of 12 November 2021 IP\_ROOT\_SERVERS = (

# IP Address "198.41.0.4",

"199.9.14.201",

"192.33.4.12",

"199.7.91.13",

"192.203.230.10",

"192.5.5.241 ",

# Name of the Root Servers # a.root-servers.net

# b.root-servers.net # c.root-servers.net # d.root-servers.net # e.root-servers.net # f.root-servers.net

"192.112.36.4", # g.root-servers.net

"198.97.190.53", # h.root-servers.net

"192.36.148.17", # i.root-servers.net

"192.58.128.30", # j.root-servers.net

"193.0.14.129", # k.root-servers.net

"199.7.83.42", # !.root-servers.net

"202.12.27.33", # m.root-servers.net

)

FORMATS=(

("CNAME", "{alias}-> alias-> {name}"),

("A", "{name}-> IPv4 address-> {address}"),

("AAAA", "{name}-> IPv6 address-> {address}"),

("MX", "{name} -> mail by -> #{preference} {exchange}"),

)

Count= 0

defResults\_Collect\_DNS(name: str, Dns\_cache: diet)-> diet:

"""

Function parses final answers into the proper data structure that print\_results requires.

'""'

Responses\_Full = {}

Domain\_Name = dos.name.from\_text(name) # Query A records

response= Dns\_look:up(Domain\_Name, dns.rdatatype.A, Dns\_cache) A=[]

for answers in response.answer:

A Rec = answers.name for answer in answers:

if answer.rdtype == 1: # A record

A.append({"name": A\_Rec, "address": str(answer)})

# Query AAAA records

response= Dns\_look:up(Domain\_Name, dns.rdatatype.AAAA, Dns\_cache) AAAA= []

for answers in response.answer: AAAA Rec = answers.name for answer in answers:

if answer.rdtype == 28: # AAAA record

AAAA.append( {"name": AAAA\_Rec, "address": str(answer)})

# Query CNAME records

response= Dns\_look:up(Domain\_Name, dns.rdatatype.CNAME, Dns\_cache) CNAME= []

for answers in response.answer: for answer in answers:

CNAME.append( {"name": answer, "alias": name})

# Query MX records

response= Dns\_lookup(Domain\_Name, dns.rdatatype.MX, Dns\_cache)

MX= []

for answers in response.answer: mx name = answers.name for answer in answers:

if answer.rdtype == 15: # MX record MX.append(

{

"name": mx\_name, "preference": answer.preference,

"exchange": str(answer.exchange),

}

)

Responses\_Full["CNAME"] = CNAME Responses\_Full["A"] = A Responses\_Full["AAAA"] = AAAA Responses\_Full["MX"] = MX

Dns\_cache.get("response\_cache")[name] = Responses\_Full return Responses\_Full

def Recurse\_Look(

Domain\_Name: dns.name.Name, qtype: dns.rdata.Rdata, ip\_, resolved, Dns\_cache: diet

) -> dns.message.Message:

"""

This function uses a recursive resolver to find the relevant answer to the query.

"""

global Count Count+= 1

outbound\_query = dns.message.make\_query(Domain\_Name, qtype) try:

response= dns.query.udp(outbound\_query, ip\_, 3) if response.answer:

resolved = True

return response, resolved elif response.additional:

if response.authority: update\_cache(response, Dns\_cache)

response, resolved = lookup\_additional(

response, Domain\_Name, qtype, resolved, Dns\_cache

)

elif response.authority and not resolved: response, resolved = lookup\_authority(

response, Domain\_Name, qtype, resolved, Dns\_cache

)

return response, resolved except Timeout:

return dns.message.Message(), False except DNSException:

return dns.message.Message(), False

def Dns\_lookup(

Domain\_Name: dns.name.Name, qtype: dns.rdata.Rdata, Dns\_cache: diet

) -> dns.message.Message:

"""

Recursive resolver has been used by a function to get the response for the query.

"""

incre = 0 Resolved= False

while incre < len(IP\_ROOT\_SERVERS): get\_Ip\_cache = ""

Name\_Find = str(Domain\_Name) next\_dot = str(Domain\_Name).find(".") while not get\_Ip\_cache and next\_dot > -1:

get\_Ip\_cache = Dns\_cache.get(Name\_Find) Name\_Find = str(Name\_Find)[next\_dot + 1 :] next\_dot = Name\_Find.find(".")

if get\_Ip\_cache:

ip\_ = get\_Ip\_cache

logging.debug("==== Found in cache ====\n")

else:

ip\_ = IP\_ROOT\_SERVERS[incre]

try:

response, Resolved = Recurse\_Look( Domain\_Name, qtype, ip\_, Resolved, Dns\_cache

)

if response.answer:

answer\_type= response.answer[0].rdtype

if qtype != dns.rdatatype.CNAME and answer\_type == dns.rdatatype.CNAME:

Domain\_Name =dns.name.from\_text(str(response.answer[0][0]))

Resolved = False logging.debug(

"--------- LOOKUP CNAME ----------- ¾s \n ¾s",

Domain\_Name, response.answer[0],

)

response= Dns\_lookup(Domain\_Name, qtype, Dns\_cache) return response

elif(

response.authority and response.authority[0].rdtype == dns.rdatatype.SOA

):

Break

else:

incre += 1 except Timeout:

incre += 1

except DNSException:

incre += 1 return response

def update\_cache(response: dns.message.Message, Dns\_cache):

"""

Function updates the cache latest results

"""

domain\_name = response.authority[O].to\_text().split(" ")[O] A\_Records = []

ITSets = response.additional for ITSet in ITSets:

for IT in ITSet:

if IT\_.rdtype == dns.rdatatype.A: A\_Records.append(str(IT\_)) Dns\_cache[domain\_name] = str(IT\_)

def lookup\_additional(response,Domain\_Name: dns.name.N ame,qtype: dns.rdata.Rdata, resolved,

Dns\_cache: diet,

):

"""

Function recursively finds additional data

"""

rrsets = response.additional

for rrset in rrsets:

for rr in rrset:

if rr\_.rdtype == dns.rdatatype.A:

response, resolved = Recurse\_Look(

Domain\_Name, qtype, str(rr\_), resolved, Dns cache

)

if resolved:

break

if resolved:

break return response, resolved

def lookup\_authority(

response,

Domain\_Name: dns.name.Name, qtype: dns.rdata.Rdata,

resolved, Dns\_cache: diet,

):

"""

Function recursively finds authority

"""

rrsets = response.authority

ns\_ip = 1111

for ITSet in ITSets:

for IT in ITSet:

if IT\_.rdtype == dns.rdatatype.NS: ns\_ip = Dns\_cache.get(str(IT\_)) if not ns\_ip:

ns\_arecords = Dns\_lookup(str(IT\_), dns.rdatatype.A, Dns\_cache)

ns\_ip = str(ns\_arecords.answer[0][0])

Dns\_cache[str(IT\_)] = ns\_ip response, resolved =

Recurse\_Look(Domain\_Name, qtype, ns\_ip, resolved, Dns\_cache)

elif IT\_.rdtype == dns.rdatatype.SOA: resolved= True

break

if resolved:

break return response, resolved

def print\_results(results: diet)-> None: """

Function takes results from Dns\_lookup, prints to the screen. """

for rtype, fmt\_str in FORMATS:

for result in results.get(rtype, []): print(fmt\_str.format(\*\*result))

def MainFn():

global Count Dns\_cache = {}

Dns\_cache["response\_cache"] = {} Args\_Parse = argparse.ArgumentParser()

Args\_Parse.add\_argument("NAME", nargs="+", help="Domain name(s) to query") Args\_Parse.add\_argument(

"-v", help="Increase the verbosity", action="store\_true) Proj\_Args = Args\_Parse.parse\_ args()

for Domain in Proj\_Args.NAME:

Count= 0

cache\_result= Dns\_cache.get("response\_cache").get(Domain) if cache result:

print\_results(cache\_result)

else:

print\_results(Results\_Collect\_DNS(Domain, Dns\_cache)) #logging.debug("Count %s", Count)

if name ==" main "·

#logging.basicConfig(level=logging.DEBUG) MainFn()

**Gui.py**

import PySimpleGUI as sg import subprocess

import sys def main():

sg.theme('Dark.Amber') #Adda touch of color # All the stuff inside your window.

layout= [ [sg.Text('Enter Domain Name to resolve:'), sg.InputText()], [sg.Button('Ok'), sg.Button('Cancel')], [sg.Output(size=(80,20))]]

# Create the Window

window= sg.Window('DNS Lookup', layout)

# Event Loop to process "events" and get the "values" of the inputs while True:

event, values = window.read()

if event== sg.WIN\_CLOSED or event== 'Cancel':# if user closes window or clicks cancel

break

print('Entered Domain Name:', values[0]) runCommand(["python","python.py", values[0]])

window.close()

def runCommand(cmd, timeout=None, window=None):

p = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)

output="

for line in p.stdout:

line= line.decode(errors='replace' if (sys.version\_info) < (3, 5) else 'backslashreplace').rstrip()

output += line print(line)

window.Refresh() if window else None # yes, a 1-line if, so shoot me retval = p.wait(timeout)

return (retval, output) # also return the output just for fun if name == ' main '·

main()