A curious case of broken DNS responses

Babak Farrokhi
RIPE 75

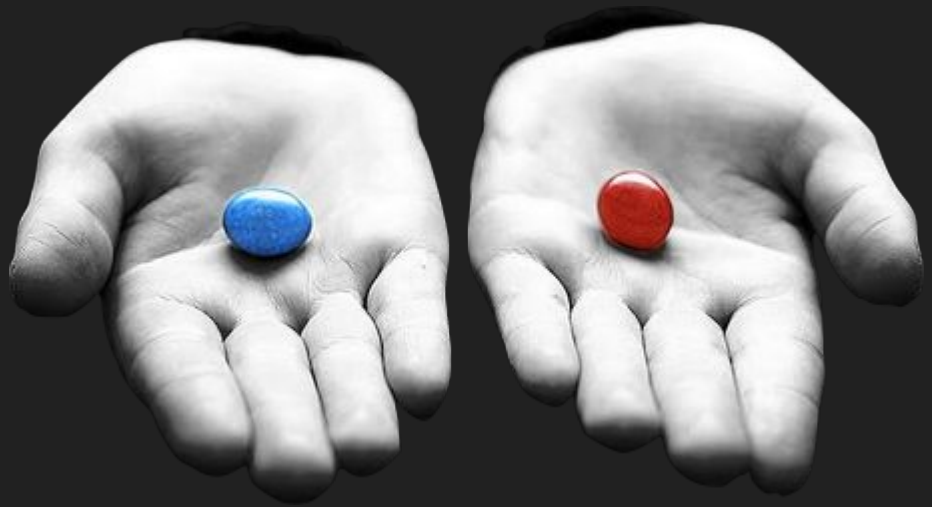
About me

- Unix SA (FreeBSD, Solaris, Linux) since 1996
- IP Networking since 1997
- FreeBSD Ports Team since 2004
- Enthusiastic coder

@farrokhi

Prologue

- When it comes to network, I always have trust issues
- Most people ignore those strange network behaviors
- Only a few people take the red pill and go down the rabbit hole...



Observation

- Outgoing SMTP fails due to MX lookup failures
  - only certain domains (e.g. twitter.com)
- Local resolver returns “incorrect” response
- Public Resolver (e.g. Google) also returned incorrect response
- I needed to look deeper into this

Down the rabbit hole...

Strange responses from public resolvers

This is not what I expected to get from a public resolver:

```
% dig +short -t A ripe.net @8.8.8.8
193.0.6.139
```

```
% dig +short -t A twitter.com @8.8.8.8
10.10.34.34
```

```
% dig -t MX twitter.com @8.8.8.8
;; Got bad packet: bad label type
45 bytes
40 10 81 80 00 01 00 01 00 00 00 00 07 74 77 69          @.............twi
74 74 65 72 03 63 6f 6d 00 00 0c 00 01 c0 0c 00          tter.com........
0c 00 01 00 00 03 79 00 03 41 41 41 00                    ......y..AAA.
```

Need to take a closer look

Not obvious at first glance, but time delta is strange...

```
% tcpdump -ttt -c2 -nqr ripe.pcap
reading from PCAP-NG file ripe.pcap
 00:00:00.000000 IP 192.168.0.132.53425 > 8.8.8.8.53: UDP, length 27
 00:00:00.138933 IP 8.8.8.8.53 > 192.168.0.132.53425: UDP, length 75
```

```
% tcpdump -ttt -c2 -nqr twitter.pcap
reading from PCAP-NG file twitter.pcap
 00:00:00.000000 IP 192.168.0.132.58418 > 8.8.8.8.53: UDP, length 29
 00:00:00.028077 IP 8.8.8.8.53 > 192.168.0.132.58418: UDP, length 45
```

dnsping - A new tool is born

- Started as a humble Python script to solve my own problem
- Not intended to re-invent the wheel
- Similar user experience as the legacy PING

```
% ./dnsping.py  -s 8.8.8.8 -c3 ripe.net
dnsping.py DNS: 8.8.8.8:53, hostname: ripe.net, rdatatype: A
32 bytes from 8.8.8.8: seq=0   time=200.371 ms
32 bytes from 8.8.8.8: seq=1   time=217.320 ms
32 bytes from 8.8.8.8: seq=2   time=236.644 ms

--- 8.8.8.8 dnsping statistics ---
3 requests transmitted, 3 responses received, 0% lost
min=200.371 ms, avg=218.112 ms, max=236.644 ms, stddev=18.149 ms
```

ICMP vs DNS Response Times

```
% ping -q -c10 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes

--- 8.8.8.8 ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 124.237/155.044/227.499/31.464 ms
```

```
% ./dnsping.py -q -s 8.8.8.8 -c3 twitter.com
dnsping.py DNS: 8.8.8.8:53, hostname: twitter.com, rdatatype: A

--- 8.8.8.8 dnsping statistics ---
3 requests transmitted, 3 responses received, 0% lost
min=12.934 ms, avg=21.355 ms, max=29.425 ms, stddev=8.251 ms
```

First anomaly

- Different domain names being treated differently
- A rogue name server impersonating as Google Resolver
- The rogue name server is close to me (given response times)
- Where is it? And how can I find out?

dnstraceroute: Traceroute tool for DNS protocol

- Similar to legacy traceroute, but for DNS protocol
- Send out actual DNS queries and expect a response
- Using TTL trick to map the journey
- Could not use legacy traceroute with UDP probes
  - The traffic redirection is based on DNS “payload”

Real vs Rogue DNS Servers

```
% ./dnstraceroute.py -s 8.8.8.8 ripe.net
dnstraceroute.py DNS: 8.8.8.8:53, hostname: ripe.net,
rdatatype: A
1   192.168.0.1 (192.168.0.1) 3.912 ms
2    *
3   192.168.10.105 (192.168.10.105) 15.792 ms
4   172.17.2.1 (172.17.2.1) 17.063 ms
5   172.17.2.9 (172.17.2.9) 11.245 ms
6   172.19.18.5 (172.19.18.5) 24.862 ms
7   172.19.17.2 (172.19.17.2) 18.972 ms
8   10.201.177.41 (10.201.177.41) 13.261 ms
9   10.10.53.190 (10.10.53.190) 14.240 ms
10  185.100.209.117 (185.100.209.117) 176.592 ms
11   *
12  de-cix.fra.google.com (80.81.192.108) 152.757 ms
13  108.170.251.193 (108.170.251.193) 90.347 ms
14  google-public-dns-a.google.com (8.8.8.8) 185.401 ms
```

```
% ./dnstraceroute.py -s 8.8.8.8 twitter.com
dnstraceroute.py DNS: 8.8.8.8:53, hostname:
twitter.com, rdatatype: A
1   192.168.0.1 (192.168.0.1) 3.160 ms
2    *
3   192.168.10.105 (192.168.10.105) 5.985 ms
4   172.17.2.1 (172.17.2.1) 8.535 ms
5   172.17.2.9 (172.17.2.9) 20.617 ms
6   172.19.18.5 (172.19.18.5) 7.823 ms
7    *
8    *
9   google-public-dns-a.google.com (8.8.8.8) 19.557 ms
```

Back to the case of broken MX

- Rogue nameserver also returns “broken” responses
- Only to certain type of queries (e.g. MX)

```
% dig -t MX twitter.com @8.8.8.8
;; Got bad packet: bad label type
45 bytes
40 10 81 80 00 01 00 01 00 00 00 00 07 74 77 69          @............twi
74 74 65 72 03 63 6f 6d 00 00 0c 00 01 c0 0c 00          tter.com........
0c 00 01 00 00 03 79 00 03 41 41 41 00                    ......y..AAA.
```

Looking at packets again...

- Response to MX request is malformed
  - Server responded with PTR response
  - `RDLENGTH` is 3 but `RDATA` field contains 4 bytes
    - Additional byte was always NULL
    - Seems like a bug in code
- Queried top 10,000 domain names [1], received 139 broken responses [2]

Uncovering the rouge resolver address

```
% dig +short -t TXT maxmind.test-ipv6.com @8.8.4.4
"ip='74.125.74.14' as='15169' isp='Google' country='FI'"
```

- Query TXT record from `maxmind.test-ipv6.com`
- It tells you the public address of your resolver
  - The address should belong to Google (AS15169)
  - Anything else means MITM
- Using RIPE Atlas to see if this is a regular practice

Is it just me? Let's ask RIPE Atlas

- 500 Probes worldwide - 484 Replied (DNS/UDP/IPv4)
  - 475 Good (Req. from Google address space) ~98%
  - 9 Bad (Req. from non-Google address space) ~2%

- Same 500 probes - 484 Replied (DNS/TCP/IPv4)
  - 479 Good ~99%
  - 5 Bad ~1%

The logic behind DNS traffic redirection

What is the motivation?

There are mainly two reasons:

1. Privacy protection (The Good)
    - Prevent sending request to use have your end-point IP address as its source address
    - Filter out malwares looking for their C&C
2. DNS based service redirection (The Evil)
    - Restrict your access
    - Redirect your traffic

Countermeasures

- Force local resolver to use TCP
- DNSCrypt (dnscrypt.org)
  - OpenDNS supports it, desktop clients available
- DNS over TLS/DTLS (RFC 7858 and 8094)
  - DNS Privacy Project (dnsprivacy.org) offers tutorials, tools, recommendations and test servers
- DNSSEC
  - From top 100 domain names only 2 of them are signed [3]
  - What if the rogue nameserver does not validate DNSSEC?

Final words

- Don’t trust a public DNS resolver, use your own
  - There ain't no such thing as a free lunch (TANSTAAFL)
  - Stub resolvers are easy to setup and use (e.g. Stubby)
- Don’t trust your upstream, encrypt as much as possible
  - DNS contains important information
  - As per RFC7258: “Pervasive Monitoring Is an Attack”

Tools of the trade

- dnsping, dnstraceroute and dnseval are part of “dnsdiag toolkit” on GitHub [4]
  - Looking for feedback and ideas from community
- Combining with other tools (e.g. RIPE Atlas) to perform more complex behavior analysis
- Suggestion: dnstraceroute in RIPE Atlas probes?

Questions?

Resources

[1] https://github.com/opendns/public-domain-lists

[2] https://gist.github.com/farrokhi/0b56ae06813391be9164

[3] https://gist.github.com/farrokhi/1d9de9df5877aaf9c42fc14412a4b0f8

[4] https://github.com/farrokhi/dnsdiag

Also my articles on RIPE Labs discussion the same issue:

- https://labs.ripe.net/Members/babak_farrokhi