Where are we in DNS privacy?

Stéphane Bortzmeyer
bortzmeyer@nic.fr
The DNS case

1. Old protocol, designed in different times
2. Leaks a lot
3. Many strong constraints: latency, reliability, ubiquity
4. First work on its privacy at CENTR, then IETF but also other efforts (DNScrypt)
Open data

An actual DNS query reveals:

1. Who is requesting (yes, I know, the status of the source IP address is complicated...)
2. What is requested (the QNAME)

It may defeat, at least partially, some security measures (such as HTTPS)
QNAME is revealing

1. www.political-party.example ← Sensitive information
2. _bittorrent-tracker._tcp.domain.example ← MPAA may be interested
3. le-pc-de-pascal.domain.example ← Personal information
4. PGP keys in DNS (indexed by user’s email, Internet-Draft just approved) ← More personal information
Who can listen?

1. Name servers (both recursors and authoritative) sysadmins. “Enablers” in RFC 6973 parlance.
2. Third-parties sniffing the cable

We need solutions for “on the wire” and “on the server”. **Encryption is not everything.**
The two principles of privacy engineering

1. Send as little data as possible (RFC 6973, section 6.1)
2. Encrypt it

1) is necessary against PRISM (or similar) providers. 2) is necessary against third-party snoopers.
Two cases

May require different solutions

1. Client machine ↔ full resolver (no caching to protect you) (you talk only to a few resolvers)
2. Resolver ↔ auth. name server (some protection because of caching and relaying by the resolver) (needs scalability)

Think how authentication is different between these two cases.
Solution? Minimizing the QNAME

1. No need to send the full QNAME to the authoritative name servers
3. Deployable unilaterally, conformant with RFCs, no change in protocol
4. Loss of data in the auth. name servers: a feature, not a bug
5. Problem with some broken name servers, such as Akamai’s
Solution? Encrypting data

1. [outside IETF] DNScurve/DNScrypt
2. IPsec (no enthusiasm)
3. A new protocol? Not Invented Here?
4. DNS over TLS. Relies on the well-known (for good and for bad) TLS. Requires TCP and therefore persistent connections
5. DNS over DTLS. Relies on the existing DTLS protocol (the same as WebRTC)
State of the project

On the standards side:

1. RFC 7626 “DNS Privacy Considerations” published
2. RFC 7816 “DNS Query Name Minimisation to Improve Privacy” published (status “experimental”)
3. RFC 7830 “The EDNS(0) Padding Option”
4. RFC 7858 “Specification for DNS over TLS” published (status “standard”)
5. A few drafts are still under discussion
Implementation of QNAME minimisation

- Unbound (version $\geq 1.5.7$). Off by default.
- Knot Resolver (version $\geq 1.0$). On by default.
server:
  qname-minimisation: yes
QNAME minimisation with Knot

dig -x of an IPv6 address, seen by tcpdump:

> 38773% [1au] NS? aRPa. (33)
> 22056% [1au] NS? Ip6.aRPa. (37)
> 43002% [1au] NS? 2.ip6.arPA. (39)
The annoying broken name servers

Knot retries with full QNAME when receiving NXDOMAIN:

< 24014*- 2/0/1 CNAME www.upenn.edu-dscg.edgesuite.net., RRSIG (270)
> 52576% [1au] NS? edGeSUItE.NET. (42)
< 52576- 0/17/15 (1034)
> 22228 [1au] NS? EdU-DScG.EdGeSUItE.nET. (51)
< 22228 NXDomain*- 0/1/1 (114)
No way to know if it is an ENT

(ENT = Empty Non-Terminal domain name) Request for www.long.verylong.detail.example:

> 19881% [1au] NS? ExaMpLE. (36) [NXDOMAIN received]
> 40708% [1au] AAAA? www.LONg.VeRylONG.DEtaIl.eXamPLE. (61)
Encrypting data

1. DNScurve/DNScrypt.
2. TLS. Relies on the well-known TLS. Main version, above TCP and therefore persistent connections (RFC 7766). Port 853.
TLS with Unbound

Implemented for a long time (1.4.22?)

ssl-service-key: "/etc/unbound/privatekeyfile.key"
ssl-service-pem: "/etc/unbound/publiccertfile.pem"
interface: 2001:db8:1::dead:beef@853
ssl-port: 853

If you don’t know OpenSSL:

openssl req -x509 -newkey rsa:4096 \ 
   -keyout privatekeyfile.key -out publiccertfile.pem \ 
   -days 1000 -nodes
Unbound starts and answers

unbound[12959:0] debug: setup TCP for SSL service
...
unbound[12959:0] debug: SSL DNS connection ip4 192.168.2.1 port 52185 (len 16)
...
unbound[12959:0] debug: Reading ssl tcp query of length 59
Client, getdns

https://getdnsapi.net/

% ./getdns/src/test/getdns/_query @192.168.2.9 -s -A -l L /www.bortzmeyer.org
...
Response code was: GOOD. Status was: At least one response was returned

(-s: stub resolver, -A: ask for addresses, -l L: TLS transport)
TLS in Go

https://miek.nl/2014/August/16/go-dns-package/

c := new(dns.Client)
c.Net = "tcp-tls"
if *insecure {
    c.TLSConfig = new(tls.Config)
    c.TLSConfig.InsecureSkipVerify = true
}
in, rtt, err := c.Exchange(m, net.JoinHostPort(ns, "853"))
The pleasures of TLS authentication

1. No auth.: vulnerable to Mallory (the man in the middle)
2. Auth.: lots of trouble ("do you really trust this expired auto-signed certificate using SHA-1?")
3. No hard rules: different profiles for authentication

```
% ./tls my-resolver internautique.fr
Error in query: x509: certificate signed by unknown authority

% ./tls -k my-resolver internautique.fr
(time 43051 µs) 2 keys. TC=false
```
See the traffic

% tshark -n -d tcp.port==853,ssl -r /tmp/dnstls.pcap
  4  0.002996 192.168.2.9 -> 192.168.2.9  SSL Client Hello
  6  0.594206 192.168.2.9 -> 192.168.2.9  TLSv1.2 Server Hello, Certificate, Server Key Exchange, Server Hello Done
  8  0.734094 192.168.2.9 -> 192.168.2.9  TLSv1.2 Client Key Exchange
 16  0.751614 192.168.2.9 -> 192.168.2.9  TLSv1.2 Application Data
 17  0.759223 192.168.2.9 -> 192.168.2.9  TLSv1.2 Application Data

(With Wireshark, Analyze → Decode as → SSL)
Combining programs

- The simplest way to deploy can be to use a proxy
- A normal resolver/cache locally, with QNAME minimisation, and encryption to an upstream server
Unbound and dnscrypt-proxy

Unbound:

server:
    interface: ::0
    qname-minimisation: yes

# To DNScrypt proxy
forward-zone:
    name: "."
    forward-addr: 127.0.0.1

dnscrypt-proxy:

dnscrypt-proxy \
    -R dnscrypt.org-fr -a 127.0.0.1 --edns-payload-size=2048
Unbound, with a public TLS resolver

https://portal.sinodun.com/wiki/display/TDNS/DNS-over-TLS+test+servers Does not check the certificate of the upstream resolver

server:  
  ssl-upstream: yes  
  tcp-upstream: yes

forward-zone:  
  name: "."  
  # To a DNS-over-TLS test server  
  # getdns team  
  forward-addr: 2a04:b900:0:100::38@853
(Provisional) Conclusion

1. We have running code
2. Deployment almost zero, currently, except for DNScrypt
Merci !