Connection-Oriented DNS to Improve Privacy and Security

Liang Zhu\textsuperscript{1}, Zi Hu\textsuperscript{1}, John Heidemann\textsuperscript{1}, Duane Wessels\textsuperscript{2}, Allison Mankin\textsuperscript{2}, Nikita Somaiya\textsuperscript{1}

\textsuperscript{1}: USC/ISI \quad \textsuperscript{2}: Verisign Labs
DNS is Essential

Example.com

192.0.2.5

DNS is simple request-response

www.example.com?

192.0.2.5

Perfect for UDP
(TCP is supported too, but as fallback and zone transfers)
Pitfall of Current DNS

DNS over UDP*  
(* except for zone transfers and fallback)

Vulnerable!

Amplification

spoofed queries appear from victim

Denial-of-Service (as victim)

DNS server

Victim (too many requests)

No Privacy!

mail.sensitive.org ?

DNS servers

Victim (too many replies)
Confront Tradition: Connection-Oriented DNS

Vulnerable!

TCP=> prevent spoofing

TLS => reduce eavesdropping

No Privacy!
Our Contributions

TCP and TLS are old, 

*proving they work acceptably* is new contribution

• Analysis of Connection-Oriented DNS
  – Client latency: only modestly more
  – Server memory: well within current hardware
• Implementation choices to get performance
• Small protocol addition: TLS upgrade
T-DNS: DNS over TCP+TLS

- Why
- Small protocol addition
- Implementation choices
- Analysis: modest cost
- Performance under attack
- Conclusion
Threat Model

- Denial of Service
- Eavesdropping
- Weak Crypto Choices
Threat Model: Denial of Service

• Problem:
  – DNS amplification attacks others
  – DoS affects server operators
    • Work-around: massive over-capacity
• Solution: TCP
  – Well understood anti-DoS methods
  – 3-way handshake preclude spoofing
  – TCP cookies shift state to client
Threat Model: Eavesdropping

• Problem: DNS privacy
  – Public DNS => more WAN traffic
  – ISPs sometimes inject ads

• Problem: DNS leaks info
  – ex: anti-spam blacklists use DNS

• Solution: TLS
  – all traffic should be private
Threat Model: Weak Crypto Choices

- Problem: policy decisions forced by UDP packet sizes
  - DNSSEC algorithm and key size
- Problem: IP fragmentation
  - Fragmentation attack\[^1\]
- What else?
  - Enough already!

T-DNS: DNS over TCP+TLS

• Why
• Small protocol addition
  – How to initiate TLS?
• Implementation choices
• Analysis: modest cost
• Performance under attack
• Conclusion
Protocol Changes: Goals

• Minimize change for easy deployment
• Reuse existing approaches
• Follow IETF patterns

implications:
- reuse TLS: Transport Layer Security
- add a STARTTLS-like “upgrade”

- Christian Huitma, IETF92: "I like this idea because it's the least innovative"
STARTTLS for DNS*

**Public**
- After TCP handshake...
  - STARTTLS/CH/TXT query with “TO” bit set
  - RCODE=0 and a TXT RR with “TO” bit set

**Private**
- www.example.com ?
- 192.0.2.5

Pros: no new port (from IANA, or in firewalls)
Cons: extra RTT Optimization?

---

*IETF Internet-Draft: draft-ietf-dprive-start-tls-for-dns-00*
Other option: port-based TLS approach
T-DNS: DNS over TCP+TLS

- Why
- Small protocol addition
- Implementation choices
  - Can we do DNS over TCP approaching UDP?
- Analysis: modest cost
- Performance under attack
- Conclusion
Connection Reuse

New connections

1. TCP 3wh
2. Start TLS
3. TLS handshake
4. Q & A

Reuse connection

1. TCP 3wh
2. Start TLS
3. TLS handshake
4. No new connection setup
5. Q & A

Fast restart

1. TCP fast open
2. Start TLS
3. TLS resumption
4. Q & A
5. 2 extra RTT
Query Pipelining

- send several queries immediately (not stop-and-wait)

*without pipelining*

1 extra RTT

q1, q2

q1

a1

q2

a2

*with pipelining*

0 extra RTT

q1, q2

q1

q2

a1

a2

pipelining matters:
62% of web has 4+ domain names
(dataset: common crawl)
Out-of-Order Processing

• Process queries on the same connection in parallel

**in-order (only)**
- (stub)
- (recursive)

**out-of-order processing**
- (stub)
- (recursive)
- (authoritative) (for Q1)
- (authoritative) (for Q2)

1 extra RTT

queries run in parallel

reply as soon as possible (maybe reorder)

out-of-order matters:
avoid head-of-line blocking
T-DNS: DNS over TCP+TLS

• Why
• Small protocol addition
• Implementation choices
• Analysis: modest cost
  – Dose connection reuse help?
  – Memory cost?
  – How about latency?
• Performance under attack
• Conclusion
What is the cost?

• Server memory
  – Is memory manageable? YES
• Client latency
  – Stub to Recursive
  – Recursive to Authoritative approach UDP
• CPU time Dominated by RTT
Suggested connection time-out: 20 s authoritative servers and 60 s elsewhere

With suggested time-out, connection reuse is high

what fraction of queries find open TCP connections?

**method:** replay 3 traces: recursive (DNSChanger, Level3) and authoritative (B-Root)

(graph shows medians, quartiles are tiny)

**conclusion:** connection reuse is often helpful
Cost of Connection Reuse? (ok!)

Connections => Memory

**method**: replay same 3 traces (here we show 2 biggest)

We apply 10kB/connection achieved by Google[2]

(experimental estimate of memory: 360kB/connection, very conservative estimate)

(graph shows medians and quartiles)

conclusion: connection reuse is often helpful and not too costly

Latency: Recursive to Authoritative

TCP and TLS vs. UDP? effects of implementation choices?

**method**: live experiments of random 140 names, each repeated 10x; recursive-authoritative RTT=35ms

(graph shows medians and quartiles for (h) and (i), or bars where median and quartiles are the same)

new connections are expensive (RTTs exactly as predicted!)
Latency: Recursive to Authoritative

TCP and TLS vs. UDP? effects of implementation choices? with long RTT (=35ms)

**method**: live experiments of random 140 names, each repeated 10x; recursive-authoritative RTT=35ms

(graph shows medians and quartiles for (h) and (i), or bars where median and quartiles are the same)

new connections are expensive (RTTs exactly as predicted!)

connections reuse can avoid much overhead

<table>
<thead>
<tr>
<th></th>
<th>median per query time (RTT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) UDP</td>
<td>1</td>
</tr>
<tr>
<td>(b) TCP</td>
<td>5</td>
</tr>
<tr>
<td>(c) TLS</td>
<td>4</td>
</tr>
<tr>
<td>(d) TCP</td>
<td>3</td>
</tr>
<tr>
<td>(e) p-TCPp-TLS</td>
<td>2</td>
</tr>
<tr>
<td>(f) p-TCPp-TLS</td>
<td>2</td>
</tr>
<tr>
<td>(g) UDP</td>
<td>2</td>
</tr>
<tr>
<td>(h) p-TCPp-TLS</td>
<td>1</td>
</tr>
<tr>
<td>(i) p-TCPp-TLS</td>
<td>1</td>
</tr>
</tbody>
</table>

connection: handshake
sending processing

-- full no reuse
stop-and-wait in-order

full reuse
stop-and-wait in-order

-- full reuse
pipeline out-of-order
End-to-End Latency: Methodology

• Many factors affect e2e performance
  – variable stub query timing
  – caching at recursive resolver
  – different RTTs (many stubs and authoritatives)

• approach: *model expected latency*
  – i.e., just averages
  – median connection reuse from trace replay
  – other parameters from experiments
End-to-End Latency: Results

Our recommendation:
- **stub-recursive: TLS**
- **recursive-authoritative: TCP**

19 to 33% slower:
modest cost -> get privacy and security

Stronger security & privacy, but more overhead
T-DNS: DNS over TCP+TLS

- Why
- Small protocol addition
- Implementation choices
- Analysis: modest cost
- Performance under attack
  - How many attackers needed to flood a server?
  - UDP vs TCP
- Conclusion
Performance Under Attack: Methodology

Controlled experiments to study different attack scenarios

Experiment Setup:

<table>
<thead>
<tr>
<th>protocol</th>
<th>attacker src IP</th>
<th>cookies</th>
<th>protocol</th>
<th>foreground resource limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>spoofed</td>
<td>n/a</td>
<td>UDP</td>
<td>CPU</td>
</tr>
<tr>
<td>TCP</td>
<td>spoofed</td>
<td>no</td>
<td>TCP</td>
<td>TCP control buffers</td>
</tr>
<tr>
<td>TCP</td>
<td>spoofed</td>
<td>yes</td>
<td>TCP</td>
<td>TCP control buffers</td>
</tr>
<tr>
<td>TCP</td>
<td>real</td>
<td>yes</td>
<td>TCP</td>
<td>TCP control buffers</td>
</tr>
</tbody>
</table>
How much Resource Needed to Flood a Server?

- Our experiments show:

<table>
<thead>
<tr>
<th>Attacker Protocol</th>
<th>Server Resource limit</th>
<th># of attackers against a single server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofed UDP</td>
<td>CPU</td>
<td>2</td>
</tr>
<tr>
<td>Spoofed TCP</td>
<td>Connections</td>
<td>N/A if SYN cookie is on</td>
</tr>
<tr>
<td>Non-sproofed TCP</td>
<td>Connections</td>
<td>6000+ with rate-limiting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 concurrent connections per source</td>
</tr>
</tbody>
</table>
Spoofed UDP DoS? Easy!

Two attackers can take down a server with more than 99% CPU usage and 230k queries/s.

Fraction of failed UDP queries:

Attack rate at server (queries/second):

Rate of 2 attackers:

USCViterbi School of Engineering Information Sciences Institute
Spoofed TCP SYN flood? Hard!

TCP SYN cookie forces the client to interact with server, ruling out spoofers.

Non-spoofed TCP: 6000+ attackers with rate limiting 10 concurrent connections per source.
T-DNS: DNS over TCP+TLS

- Why
- Small protocol addition
- Implementation choices
- Analysis: modest cost
- Performance under attack
- Conclusion
Conclusion

• DNS over TCP and TLS to improve security and privacy
• Our analysis shows performance is acceptable
• Join us to Improve DNS at IETF
  – draft-ietf-dprise-start-tls-for-dns-00
  – draft-ietf-dnsop-5966bis-01
• Try T-DNS:
  – http://www.isi.edu/ant/software/tdns
• Feedback?