Enabling Privacy-Aware Zone Exchanges Among Authoritative and Recursive DNS Servers

Nikos Kostopoulos, Dimitris Kalogerias and Vasilis Maglaris

NETwork Management & Optimal Design (NETMODE) Laboratory
School of Electrical & Computer Engineering
National Technical University of Athens (NTUA)
Motivation: DNS Water Torture Attacks

- DDoS attacks can be mitigated more efficiently close to their origins

**Our use case for DNS: Scrubbing services, Recursive DNS Server Filters**

- However, **AXFR** requests are typically restricted for security reasons
Contribution

- A privacy-aware schema for the efficient distribution of Authoritative DNS Server zones to Recursive DNS Servers or scrubbing services

**Design Requirements:**

→ **Privacy-aware zone distribution**
→ **Efficient zone mapping** (storage, filtering latency, consumed bandwidth)
→ **Compatibility with the existing DNS infrastructure** (AXFR, IXFR requests)
→ **Support for incremental updates**

- Relying on *probabilistic data structures* as datastores for valid Authoritative DNS Server zone names. These fulfill the previous design requirements.

- Extending previous work (*IEEE CloudNet 2019*): *Bloom Filters* were used to map the names of large DNS zones and filter suspicious DNS traffic in cloud infrastructures
  → In this paper, we implement the zone distribution mechanism
  → Instead of *Bloom Filters*, we use *Cuckoo Filters* that support item deletion
Background: Bloom Filters

- **Bitarrays** (of \( m \) bits) used for Approximate Membership Lookups:
  
  Is element \( x \) stored in the *Bloom Filter*?

- All bits are initially set to 0.
  
  Each element is hashed with \( k \) different *hash functions*.
  Corresponding positions (\( \text{hash results mod } m \)) are set to 1.

\[
\begin{array}{lcccccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}
\]

\( m \) bits

\( k = 3 \)

word1

word2

Bits may be shared by multiple items

False Negatives (Item in the filter, lookup says it is not): **Impossible**
False Positives (Item not in the filter, lookup says it is): **Possible**
Bloom Filter based Approaches for DNS

- Related approaches:
  - Mapping DNSSEC zone names to accelerate authenticated responses
  - Logging DNS data
  - Detecting botnet traffic
  - Tracking newly observed domain names

Privacy-aware approaches, but deletions are not supported

Cuckoo Filters vs Bloom Filters:
  → Cuckoo Filters are more time and space efficient
  → Cuckoo Filters support element deletion
### Background: Cuckoo Filters

- Elements are inserted as fingerprints in entries of a 2D array
  - Fingerprints of size $f$ bits are calculated using the function $fgp()$

**Cuckoo Filters are characterized by:**
- Number of available buckets $m$
- Fingerprint entries $b$ per bucket

Each element $x$ is assigned a pair of buckets $h_1$ and $h_2$:

<table>
<thead>
<tr>
<th>$h_1(x)$</th>
<th>$h_2(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$hash(x)$</td>
<td>$h_1(x) \oplus hash(fgp(x))$</td>
</tr>
</tbody>
</table>

**Example for $m=4$, $b=2$:**

#### Inserting $x'$ fingerprint 2 times

- $h_1(x)$
- $h_2(x)$
- $fgp(x)
- $fgp(x)$

One of the two buckets is randomly selected.

#### Inserting $y'$ fingerprint

- $h_1(y)$
- $h_2(y)$
- $fgp(x)$
- $fgp(x)$
- $fgp(y)$

$fgp(x)$ evicted to alternate bucket.

$x$ and $y$ share a bucket.
Baseline Design

- **Privacy-Aware Zone Manager**

- **Hashed DNS Zones**

- **Incremental DNS Zones**
Implementation: The Privacy-Aware Zone Manager

Builds and maintains the Cuckoo Filters whose fingerprints are used to create and revise the privacy-aware DNS zones

Actions:
- Retrieves Plaintext DNS Zone RR’s, hashes their FQDN into fingerprints, creates Cuckoo Filters and the Hashed DNS Zones
- Retrieves Plaintext DNS Zone changes regularly, updates the in-memory Cuckoo Filters and the Incremental DNS Zones
- Ignores RR’s whose value was updated, but their FQDN did not change
- Special treatment for RR’s that share FQDN’s with others, but differ in RR type and/or value (usage of frequency counters)

- Implemented in Python 3
- Murmurhash3 for fingerprint and hash calculations
Implementation: Hashed DNS Zones (1)

These zones hold the *FQDN’s* of the *Plaintext DNS Zones* hashed and mapped in *Cuckoo Filters* (*Use of AXFR*)

**Serialization format (zone *hszn.tld*):**

1: ; Zone: *hszn.tld*
2: ; Cuckoo Filter Parameters
3: buckets.hszn.tld IN TXT <m>
4: entries.hszn.tld IN TXT <b>
5: fgp-size.hszn.tld IN TXT <f>
6: fgp-algo.hszn.tld IN TXT <fgp()>
7: hash-algo.hszn.tld IN TXT <hash()>
8: ; Cuckoo Filter Data
9: <n>.hszn.tld IN TXT <RR Data>

**Cuckoo Filter parameters & algorithms:**

- Number of buckets *m*, fingerprint size *f*, number of entries *b*
- Algorithms used for fingerprint and candidate buckets calculation
Example for the 1st data RR of the .ntua.gr Hashed DNS Zone

Cuckoo Filter with:
- $f=12$ bit fingerprints
- $b=4$ entries / bucket
- 82 fingerprints mapped

Rules:
- Equally sized fingerprints of $\lceil f/4 \rceil$ Bytes (hex digits).
- Fingerprints requiring less than $\lceil f/4 \rceil$ Bytes are prepended with 0’s
- The fingerprints of multiple Cuckoo Filter buckets are mapped sequentially within a single TXT type RR
- Buckets with vacant entries require a trailing dot as they do not have explicit boundaries. Full buckets do not.
- TXT type RR limit: 255 Bytes
They map name changes of *Plaintext DNS Zones* (*Use of IXFR*)

**Serialization format** (*zone inczn.tld*):

1. `; Zone: inczn.tld`
2. `; Zone Parameters`
3. `last-serial.inczn.tld` IN TXT `<serial>`
4. `sequence.inczn.tld` IN TXT `<seq-no>`
5. `; Updates`
6. `<n>.inczn.tld` IN TXT " `<fgp> <action> <h1>,<h2>`"

**Rules:**

- **last-serial:** Changes prior to this value are incorporated in the *Hashed DNS Zones*. Starting point for *Recursive DNS Servers* to begin retrieving data from an *Incremental DNS Zone*

- **sequence:** Defines if a *Hashed DNS Zone* is stale and must be downloaded again, e.g. when *Cuckoo Filter* parameters change

- **Updates:** The fingerprint of the name that changed, action (name added/deleted) and buckets of the fingerprint in the *Cuckoo Filter*
Evaluation: Testbed & Dataset

Testbed:
- **Authoritative DNS Server:** VM with 2 vCPUs, 16 GB RAM
- **DNS Software:** BIND9

Available DNS Zones:
- **.ntua.gr:** 8,294 distinct FQDN’s
- **.su:** 109,719 distinct FQDN’s
- **.se:** 1,387,690 distinct FQDN’s
- **.ru:** 5,325,231 distinct FQDN’s
Hashed DNS Zones Privacy-Awareness

Cuckoo Filters store names hashed, but attackers may attempt to gain insight into zone contents by performing brute force attacks.

**Target:** Assess the capabilities of Cuckoo Filters to withstand brute force attacks in the context of DNS.

Evaluation of *True Positives* (TP’s) and *False Positives* (FP’s) looking up all permitted name combinations with 1st label length of 3-7 chars.

<table>
<thead>
<tr>
<th>1st Label Length (Characters)</th>
<th>TP’s (FQDN’s)</th>
<th>FP’s (FQDN’s)</th>
<th>FP’s/TP’s (Ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>320</td>
<td>57</td>
<td>0.18</td>
</tr>
<tr>
<td>4</td>
<td>640</td>
<td>1,789</td>
<td>2.80</td>
</tr>
<tr>
<td>5</td>
<td>1,178</td>
<td>68,296</td>
<td>57.98</td>
</tr>
<tr>
<td>6</td>
<td>1,183</td>
<td>2,532,293</td>
<td>2,140.57</td>
</tr>
<tr>
<td>7</td>
<td>1,363</td>
<td>93,665,989</td>
<td>68,720.46</td>
</tr>
</tbody>
</table>

- Zone: ntua.gr
- FP ratio: 0.3%
- 37 possible characters (letters, digits, hyphen)

- FQDN’s with 1st label longer than 5 chars protected with high certainty
- Longer 1st labels result into more False Positives
Hashed DNS Zones Serialization

Target: Determine the applicability of diverse data serialization formats for mapping zone names into *Hashed DNS Zones*

Considered serialization formats:
- *Cuckoo Filter* with multiple buckets mapped within each *RR*
- *Cuckoo Filter* with a single bucket mapped within each *RR*
- *Bloom Filter* with multiple Bytes mapped within each *RR*

Bandwidth consumption during an AXFR request:

<table>
<thead>
<tr>
<th>Indicative Zone (Distinct FQDN’s)</th>
<th>Information Serialization Format</th>
<th>Cuckoo Filters (Actual Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cuckoo Filter (Multiple Buckets / RR)</td>
<td></td>
</tr>
<tr>
<td>ntua.gr (8,294)</td>
<td>26.77 KB</td>
<td>13.51 KB</td>
</tr>
<tr>
<td>su (109,719)</td>
<td>352.1 KB</td>
<td>178.58 KB</td>
</tr>
<tr>
<td>se (1,387,690)</td>
<td>4.36 MB</td>
<td>2.21 MB</td>
</tr>
<tr>
<td>ru (5,325,231)</td>
<td>16.78 MB</td>
<td>8.46 MB</td>
</tr>
<tr>
<td></td>
<td>Cuckoo Filter (Single Bucket / RR)</td>
<td></td>
</tr>
<tr>
<td>ntua.gr (8,294)</td>
<td>63.91 KB</td>
<td></td>
</tr>
<tr>
<td>su (109,719)</td>
<td>876.1 KB</td>
<td></td>
</tr>
<tr>
<td>se (1,387,690)</td>
<td>11.21 MB</td>
<td></td>
</tr>
<tr>
<td>ru (5,325,231)</td>
<td>43.76 MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bloom Filter (Multiple Bytes / RR)</td>
<td></td>
</tr>
<tr>
<td>ntua.gr (8,294)</td>
<td>41.86 KB</td>
<td></td>
</tr>
<tr>
<td>su (109,719)</td>
<td>553.11 KB</td>
<td></td>
</tr>
<tr>
<td>se (1,387,690)</td>
<td>6.86 MB</td>
<td></td>
</tr>
<tr>
<td>ru (5,325,231)</td>
<td>26.34 MB</td>
<td></td>
</tr>
</tbody>
</table>

The *Cuckoo Filter* with multiple buckets/RR format outperforms the others
Hashed DNS Zones Management

Target: Latency comparison of actions related to managing the Hashed DNS Zones using both Bloom Filters and Cuckoo Filters

Actions:
- Initial creation of the Hashed DNS Zones in memory (.ru zone)
- Updating the data structures (1,000 deletions, 1,000 insertions)

- Bloom Filters are created faster than Cuckoo Filters due to the element eviction process of Cuckoo Filter insertions (single time action)
- Cuckoo Filters rapidly incorporate changes (Bloom Filters are rebuilt)
Conclusion & Future Work

Our approach is promising for distributing Authoritative DNS Server zone names efficiently, while preserving privacy

Future Work:

- Investigate recently proposed probabilistic data structures, e.g. Morton Filters, Xor Filters and Vacuum Filters
- Employ data plane programming to protect the open channel used for relaying zone exchanges (XDP)
- Adapt solution to the mitigation of amplification NXNSAttacks
- Develop a Distributed and Federated Learning detection mechanism that will reduce our zone sizes by excluding infrequently requested names
Enabling Privacy-Aware Zone Exchanges Among Authoritative and Recursive DNS Servers

Open-Sourced Code: https://github.com/nkostopoulos/dnspriv

Contact Details: nkostopoulos@netmode.ntua.gr

THANK YOU!