Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers

Audrey Randall, Enze “Alex” Liu, Gautam Akiwate, Ramakrishna Padmanabhan, Geoffrey M. Voelker, Stefan Savage, Aaron Schulman
Harmful Internet behavior today

Common Internet abuse (well studied)

- Spam Emails
- Botnets
- Malware

Rare Internet abuse (sparsely studied)

- Hack for Hire
- Stalkerware
- Typo Squatting
- www.googlw.it
Harmful Internet behavior today

Common Internet abuse (well studied)
- Spam Emails
- Botnets
- Malware

Rare Internet abuse (sparsely studied)
- Typo Squatting
- Hack for Hire
- Stalkerware

Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers
Categories of harmful Internet behavior

- Hack for Hire
- Stalkerware
- Typo Squatting

If you can observe DNS requests, you can study these types of harm.

Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers
New Era in DNS: Public Resolvers

Public resolvers are gaining popularity. They’re now often used by default!
- Google home routers
- Firefox
- NYC Public WiFi

Can a third-party observer use these services to observe rare behavior?

Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers
Observing requests on public resolvers

Well-known technique: DNS cache snooping.

In the past, considered a privacy threat.
  • Often used misconfigured home routers

Public DNS resolvers allow preserving privacy!
  • Too many users to de-anonymize

But, public resolvers are more challenging...
  • Complicated caching strategies -> some protocol noncompliance
Organization of this talk

1. Background on cache snooping
2. Reverse engineering public resolver caching strategies
3. Our tool: Trufflehunter
4. Case studies
Organization of this talk

1. Background on cache snooping
2. Reverse engineering public resolver caching strategies
3. Our tool: Trufflehunter
4. Case studies
Background: How Cache Snooping Works

User asks example.com?

If example.com is present?

If cache snooper catches it?

If authoritative nameserver exists?

Example: example.com 1.2.3.4 TTL=60
Background: How Cache Snooping Works

User queries DNS resolver with example.com?
RD = False

Cache snooper intercepts the query and responds with:
example.com 1.2.3.4 TTL=30

Response is cached for 30 seconds.
Background: How Cache Snooping Works

Cache snooping provides a lower bound on the number of users accessing a domain.
Organization of this talk

1. Background on cache snooping
2. Reverse engineering public resolver caching strategies
3. Our tool: Trufflehunter
4. Case studies
Simplified Public Resolver Cache Architecture
Public resolvers use novel caching algorithms

Each resolver implements caching differently
- Inconsistency causes potential problems
- Some algorithms cause TTL violations

To count filled caches, must identify which caches queries hit!

We reverse-engineered each caching strategy.
- Used only TTL, timestamp
How We Modeled Cache Architectures

Experiment:

1. Repeatedly query resolver, fill caches
2. Observe how queries were cached: examine TTLs.

“TTL Line:” Model of how a TTL decreases in a cache.

- Rate: one second per second.

![Graph showing TTL decay over time.](image)
OpenDNS and Quad9
OpenDNS and Quad9
OpenDNS and Quad9

[Graph showing TTL (secs) on the y-axis and Time (secs) on the x-axis, with data points indicating a decay over time.]

[Diagram illustrating the flow of data between a Frontend and a Backend system, with labels such as 575, 573, 570, 560, 558, 556, 554, 552, and 550.]
Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers
Cloudflare

Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers
Cloudflare

![Graph showing TTL (secs) over Time (secs)]

Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers
Cloudflare
Does Cloudflare’s strategy lead to inaccurate TTLs?

Max drift we saw: ~80s (TTL=3hrs)

Drift scales with max TTL, so problems likely to be minimal?
And then there’s Google DNS...

Prior work observed Google “mystery caches”

• Schomp et al. found initial TTL correct, subsequent TTLs often incorrect
• Rohprimardho et al.: “Ghost caches”

Why are caches getting filled without being queried?
Google DNS

![Graph showing TTL (secs) vs Time (secs) with data points and trend line.](image)
Google DNS

No measurements!
Google DNS

The graph shows the TTL (Time To Live) of a domain over time. The x-axis represents time in seconds, ranging from 0 to 50. The y-axis represents the TTL, ranging from 600 to 540 seconds. The data points are plotted and connected with a line, indicating the decay of TTL over time. The graph includes data points at times 597, 575, 564, 551, and 549 seconds.
Google DNS: Dynamic Caching

User receives backend TTL (550)

Frontend cache stores max TTL (600)
Google DNS: Dynamic Caching

Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers
Does Google’s strategy lead to inaccurate TTLs?

“Extra” front-end caches cleared when backend TTL expires.

Maximum drift: $2 \times \text{(max TTL)}$.

Question: Why store max TTL in frontend caches?
Summary of caching strategies
Organization of this talk

1. Background on cache snooping
2. Reverse engineering public resolver caching strategies
3. Our tool: Trufflehunter
4. Case studies
Trufflehunter

Distributed measurement tool
  • Deployed on CAIDA’s Ark project

Sends DNS queries across the U.S.

Interprets the responses, estimate counts of users

Three months of data: March 6 – May 29 2020
How accurate is Trufflehunter at estimating filled caches?

Experiment:
- Place domain we control into caches
- Observe it with Trufflehunter
- Requests to our authoritative nameserver = true number of filled caches

Error in number of filled caches:
Bounds on Observable Users

(Cloudflare has only one visible cache per PoP.)
Organization of this talk

1. Background on cache snooping
2. Reverse engineering public resolver caching strategies
3. Our tool: Trufflehunter
4. Case studies
Case Studies

Three case studies:
  • Stalkerware
  • Contract Cheating
  • Typo Squatting

Previously, all were hard to measure – little data available about prevalence.
Case Study #1: Stalkerware

Stalkerware: emerging spyware threat.
  • Often records location, keyboard, ambient sound/video
  • Can hide its presence

We download and profile 24 apps
  • 6 dual use: Usually marketed for parental control, employee surveillance.
  • 16 overt: “Undetectable”
  • Record network traffic: extract DNS requests
Why is stalkerware hard to study by other means?

Prior work: clinical settings
  • Individual one-on-one sessions: low sample size
  • Few to zero overt apps found in the wild

Targets have often already reset devices

Clinics often lack technical expertise
From Counting Caches to Counting Devices

Stalkerware often makes DNS requests automatically, at regular intervals.

\[
\text{Devices with stalkerware installed} = \frac{\text{Filled Caches}}{\text{App Request Rate}}
\]
At least 5,700 people are targeted by overt stalkerware in the U.S. today.
Observed Stalkerware Dashboard Visits

Popularity of app ≠ popularity of dashboard
Case Study #2: Contract Cheating

Get better grades, effortlessly.

- Services complete homework, projects, even entire classes
- Hard to detect – original content, plagiarism checkers don’t work

Trufflehunter: Cache Snooping Rare Domains at Large Public DNS Resolvers
Observed Contract Cheating

Some services decrease over time: schools letting out for summer break?
Case Study #3: Typo Squatting

Even though domains are old and probably blacklisted, we see requests.
Takeaway: Don’t get rid of cache snooping yet!

Minimal privacy concerns on public resolvers
  • Too many users to de-anonymize

Can measure types of harm that are otherwise difficult to study
  • Stalkerware
  • Contract cheating
  • New phenomena
    • Hack-for-hire services
    • Phishing
Conclusion

Public DNS resolvers enable **privacy-preserving cache snooping**
  • Valuable measurement technique – should not be disabled

Public resolver cache architecture is complex
  • We reverse engineer four resolvers’ strategies
  • Cloudflare, Google cause **minor TTL noncompliance**

We found **non-trivial lower bounds** of the prevalence of hard-to-study Internet phenomena.

https://github.com/ucsdsysnet/trufflehunter