DNS-Based User Tracking

Amit Klein

Joint research with Benny Pinkas
Why do we need user tracking?

From the literature:

• Real-time targeted marketing
• Campaign measurement
• Fraud detection
• Protection against account hijacking
• Anti-bot and anti-scraping services
• Enterprise security management
• Protection against DDOS attacks
• Reaching customers across devices
• Limiting number of accesses to services
User tracking in 1999...

• Cookies!
• Later, also: localStorage and friends
• Two browsers (IE+Mozilla), one OS (Windows)
User tracking in 2019 – the challenges

- **Privacy mode** boundary
- Identical HW+SW (the “golden image” problem)
- **Many browsers** (IE, FF, GC, Safari) on desktops/laptops (Windows, macOS) and mobile (iOS, Android)
- **Awareness** – history clearing, browser restart, browser per task
Have the cake and eat it too

<table>
<thead>
<tr>
<th></th>
<th>Fingerprinting (typical)</th>
<th>Tagging (typical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privacy mode boundary</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Identical HW+SW</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Coverage</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>History cleanup</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Browser restart</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cross-browser</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
### Have the cake and eat it too

<table>
<thead>
<tr>
<th></th>
<th>Fingerprinting (typical)</th>
<th>Tagging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privacy mode boundary</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Identical HW+SW</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Coverage</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>History cleanup</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Browser restart</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cross-browser</td>
<td>✓</td>
<td>x</td>
</tr>
</tbody>
</table>

We designed a scheme that satisfies all six requirements.
Have the cake and eat it too

• We devised a technique that basically satisfies all 6 requirements
• DNS-based (duh)
• Some disclaimers:
  • Good coverage (resolver SW), but not perfect
  • Cross browser works, but not in some browser combinations
  • Doesn’t work across network switches (and OS restart)
  • TTL limitations
DNS refresher

Client (OS)

Browser
Navigate to http://www.example.com/

Stub resolver
getaddrinfo www.example.com
www.example.com → 10.1.1.1

Stub resolver cache

Resolution platform
A? www.example.com
www.example.com A 10.1.1.1

Resolution platform cache

Web server (10.1.1.1)

Root N.S.

.com N.S.

.example.com N.S.

Navigate to http://www.example.com/
getaddrinfo www.example.com
www.example.com A 10.1.1.1
www.example.com → 10.1.1.1
DNS-based user tracking

WHAT IF I TOLD YOU

I CAN MAKE EACH USER GET A DIFFERENT COMBINATION OF DNS RESOLUTIONS
The main idea (example)

- User 1:
  - $x_1.anonymity.fail \rightarrow 10.4.5.6, \ldots$ (2)
  - $x_2.anonymity.fail \rightarrow 10.1.2.3, \ldots$ (1)
  - $x_3.anonymity.fail \rightarrow 10.7.8.9, \ldots$ (3)
  - $\ldots$
    - ID=$(2,1,3,\ldots)$

- User 2:
  - $x_1.anonymity.fail \rightarrow 10.1.2.3, \ldots$ (1)
  - $x_2.anonymity.fail \rightarrow 10.1.2.3, \ldots$ (1)
  - $x_3.anonymity.fail \rightarrow 10.4.5.6, \ldots$ (2)
  - $\ldots$
    - ID=$(1,1,2,\ldots)$
The main idea - components

• Tracking is carried out via an HTML+Javascript “snippet” which you can place in any page.
• The snippet references Javascript resources on multiple hosts \((x_1,\ldots,x_N)\) in the tracking domain (managed via a dedicated auth. name server).
• The tracker also runs a web server farm. Each web server \(j\) has a dedicated IP address and returns a different JS code (e.g. \(v[i]=j\))
The main idea - technique

Client (OS)

Browser

Tracking snippet:

<script src="x_i,...">:

Stub resolver

Resolution platform

Stub resolver cache

Resolution platform cache

Root N.S.

.fail N.S.

.anonymity.fail N.S.

Tracking web servers

A 10.1.2.3

A 10.7.8.9

A 10.4.5.6

x_i,... →

10.4.5.6

10.7.8.9

10.1.2.3

x_i,...

A 10.4.5.6

A 10.7.8.9

A 10.1.2.3

v=1

v=2

v=3

GET / HTTP/2.1

Host: x_i,...
Mandatory requirements

• **Req #1**: Same client must get same ID each time (for a reasonable time)
  • Caching at the Stub Resolver ensures this
• **Req #2**: Different clients must get different IDs
  • This is obvious for clients that use different DNS resolvers (each resolver gets its own order of IPs)
  • But what happens with clients behind the same resolver?
IDs in the same farm

• Main problem: the answer (list of IP addresses) is cached in the resolver itself!

• So theoretically, the resolver returns the same response to all its clients (and they all get the same ID). Right?
  • Not necessarily. BIND 9.x (the most popular SW) randomizes the order!
  • Microsoft DNS server, MaraDNS do round robin – we can still use this.
  • Unbound, PowerDNS – fixed order (bad). But a very small portion of the landscape.
IDs in the same farm – multiple resolvers

• Load-balanced “farm” of resolvers works in the tracker’s favor!
• Clients are load balanced over resolvers, so even if a single resolver does return data in the same order, load balancing among resolvers provides the necessary randomness
Complications and limitations

- **Windows: dual cache**: IE/Edge+Firefox, vs. Chrome+Opera
- **macOS: Chrome has its own stub resolver** (but Safari and Firefox share the stub resolver cache)
- **TTL cap** – most resolvers put a cap on the TTL (7d-¼d), stub resolvers as well.
- **Disconnecting** from the network automatically flushes the stub resolver DNS cache
- **Restarting** the machine flushes the DNS cache
How do we score?

• Privacy mode boundary – **GOOD**. Both modes use the stub resolver cache.
• Identical HW+SW – **GOOD**. Each device gets a random ID.
• Coverage – **PRETTY GOOD**. Except for single Unboud resolver or single PowerDNS>3.6 resolver. Coverage >90% for enterprises.
• History cleanup – **GOOD**. Doesn’t touch the stub resolver cache (except Chrome on macOS).
• Browser restart – **GOOD**. Ditto.
• Cross browser – **GOOD**. Except Chrome on macOS, and the dual cache on Windows.
Mitigations

• Systematic solution (need both):
  • Browsers use random IP from RRset for each new connection
    • Takes care of the “randomized” RRset approach (|RRset|>1)
  • Sticky-by-client (IP) DNS load balancing
    • Takes care of the load-balancing approach with |RRset|=1 (there’ll be only |resolvers| possible IDs)

• Forward shared HTTP proxy (or Tor)
• Flush DNS cache very often
• Tracking domain blacklisting (cat and mouse)
Conclusions

• A new user tracking method:
  • DNS Based
  • Crosses the privacy mode boundary
  • Handles the golden image challenge
  • Has good coverage

• Not easy to mitigate!

• Additional results (non-DNS-tracking):
  • DNS load balancing strategies (good for connecting to a specific resolver)
  • Systematic info about resolver SW, stub resolver SW, browser DNS behavior
Thanks!