Mitigating Browser Fingerprinting in Web Specifications

W3C Editor's Draft 21 May 2018

This version:
https://w3c.github.io/fingerprinting-guidance/

Latest published version:
https://www.w3.org/TR/fingerprinting-guidance/

Latest editor's draft:
https://w3c.github.io/fingerprinting-guidance/

Editor:
Nick Doty

Version history:
GitHub commit history

Issues list:
GitHub issues list

Abstract

Exposure of settings and characteristics of browsers can harm user privacy by allowing for browser fingerprinting. This document defines different types of fingerprinting, considers distinct levels of mitigation for the related privacy risks and provides guidance for Web specification authors on how to balance these concerns when designing new Web features.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This document is a draft Interest Group Note to provide guidance to Web specification authors on mitigating the privacy impacts of browser fingerprinting, currently under development by the Privacy Interest Group (PING). A snapshot draft of this Note was published on 24 November

Table of Contents

1. Browser fingerprinting

1.1 What is fingerprinting?

In short, ***browser fingerprinting*** is the capability of a site to identify or re-identify a visiting user, user agent or device via configuration settings or other observable characteristics.

A similar definition is provided by [RFC6973]. A more detailed list of types of fingerprinting is included below. This document does not attempt to catalog all features currently used or usable for browser fingerprinting; however, A. Research provides links to browser vendor pages and academic findings.

1.2 Privacy impacts and threat models

Browser fingerprinting can be used as a security measure (e.g. as means of authenticating the user). However, fingerprinting is also a potential threat to users' privacy on the Web. This document does not attempt to provide a single unifying definition of "privacy" or "personal data", but we highlight how browser fingerprinting might impact users' privacy. For example, browser fingerprinting can be used to:

- identify a user
- correlate a user’s browsing activity within and across sessions
- track users without transparency or control

The privacy implications associated with each use case are discussed below. Following from the practice of security threat model analysis, we note that there are distinct models of privacy threats for fingerprinting. Defenses against these threats differ, depending on the particular privacy implication and the threat model of the user.

1.2.1 Identify a user

There are many reasons why users might wish to remain anonymous or unidentified online, including: concerns about surveillance, personal physical safety, concerns about discrimination against them based on what they read or write when using the Web. When a browser fingerprint is correlated with identifying information (like an email address or legal name), an application or service provider may be able to identify an otherwise pseudonymous user.

Users concerned about physical safety from, for example, a governmental adversary might employ onion routing systems such as Tor to limit network-level linkability but still face the danger of browser fingerprinting to correlate their Web-based activity.

> NOTE
>
> Is this privacy implication usefully distinct from unexpected correlation? How does this relate to linkability to other identities? [via TAG feedback]

1.2.2 Correlation of browsing activity

Browser fingerprinting raises privacy concerns even when real-world identities are not implicated. Some users may be surprised or concerned that an online party can correlate multiple visits (on the same or different sites) to develop a profile or history of the user. This concern may be heightened because (see below) it may occur without the user's knowledge or consent and tools such as clearing cookies do not prevent further correlation.

Browser fingerprinting also allows for tracking across origins [RFC6454]: different sites may be able to combine information about a single user even where a cookie policy would block accessing of cookies between origins, because the fingerprint is relatively unique and the same for all origins.

1.2.3 Tracking without transparency or user control

In contrast to other mechanisms defined by Web standards for maintaining state (e.g. cookies), browser fingerprinting allows for collection of data about user activity without clear indications that such collection is happening. Transparency can be important for end users, to understand how ongoing collection is happening, but it also enables researchers, policymakers and others to document or regulate privacy-sensitive activity. Browser fingerprinting also allows for tracking of activity without clear or effective user controls: a browser fingerprint typically cannot be cleared or re-set. (See the finding on unsanctioned tracking [TAG-UNSANCTIONED].)

1.3 What can we do about it?

Advances in techniques for browser fingerprinting (see A. Research, below), particularly in active fingerprinting, suggest that elimination of the capability of browser fingerprinting by a determined adversary through solely technical means that are widely deployed is implausible. However, mitigations in our technical specifications are possible, as described below (6. Mitigations), and may achieve different levels of success (4. Feasibility).

Mitigations recommended here are simply mitigations, not solutions. Users of the Web cannot confidently rely on sites being completely unable to correlate traffic, especially when executing client-side code. A fingerprinting surface extends across all implemented Web features for a particular user agent, and even to other layers of the stack; for example, differences in TCP connections. In order to mitigate the risk as a whole, fingerprinting must be considered during the design and development of all specifications.

The TAG finding on Unsanctioned Web Tracking, including browser fingerprinting, includes description of the limitations of technical measures and encourages minimizing and documenting new fingerprinting surface [TAG-UNSANCTIONED]. The best practices below detail common actions that authors of specifications for Web features can take to mitigate the privacy impacts of browser fingerprinting.

2. Best Practices Summary

- Best Practice 1: Avoid unnecessary or severe increases to fingerprinting surface.
- Best Practice 2: Mark features that contribute to fingerprintability.
- Best Practice 3: Specify orderings and non-functional differences.
- Best Practice 4: Design APIs to access only the entropy necessary.
- Best Practice 5: Enable graceful degradation for privacy-conscious users or implementers.
- Best Practice 6: Avoid unnecessary new local state mechanisms.
- Best Practice 7: Highlight any local state mechanisms to enable simultaneous clearing.
- Best Practice 8: Limit permanent or persistent state.

3. Types of fingerprinting

3.1 Passive

Passive fingerprinting is browser fingerprinting based on characteristics observable in the contents of Web requests, without the use of any code executed on the client.

Passive fingerprinting would trivially include cookies (often unique identifiers sent in HTTP requests), the set of HTTP request headers and the IP address and other network-level information. The User-Agent string, for example, is an HTTP request header that typically identifies the browser, renderer, version and operating system. For some populations, the User-Agent string and IP address will commonly uniquely identify a particular user's browser [NDSS-FINGERPRINTING].

3.2 Active

For ***active fingerprinting***, we also consider techniques where a site runs JavaScript or other code on the local client to observe additional characteristics about the browser. Techniques for active fingerprinting might include accessing the window size, enumerating fonts or plug-ins, evaluating performance characteristics, or rendering graphical patterns. Key to this distinction is that active fingerprinting takes place in a way that is potentially detectable on the client.

3.3 Cookie-like

Users, user agents and devices may also be re-identified by a site that first sets and later retrieves state stored by a user agent or device. This ***cookie-like fingerprinting*** allows re-identification of a user or inferences about a user in the same way that HTTP cookies allow state management for the stateless HTTP protocol [RFC6265].

Cookie-like fingerprinting can also circumvent user attempts to limit or clear cookies stored by the user agent, as demonstrated by the "evercookie" implementation [EVERCOOKIE]. Where state is maintained across user agents (as in the case of common plugins with local storage), across devices (as in the case of certain browser syncing mechanisms) or across software upgrades, cookie-like fingerprinting can allow re-identification of users, user agents or devices where active and passive fingerprinting might not.

4. Feasibility

4.1 Fingerprinting mitigation levels of success

There are different levels of success in addressing browser fingerprinting:

Decreased fingerprinting surface
Removing the source of entropy or accessible attributes that can be used for fingerprinting.

Increased anonymity set
By standardization, convention or common implementation, increasing the commonality of particular configurations to decrease the likelihood of unique fingerprintability.

Detectable fingerprinting
Making (in particular, active) fingerprinting observable to others, so that the user agent might block it or researchers can determine that it's happening.

Clearable local state
Helping users respond to fingerprinting by making state mechanisms clearable.

4.2 Feasible goals for specification authors

This document works under the expectation that mitigations with different levels of success are feasible under different circumstances, for different threat models and against different types of fingerprinting. In general, active fingerprinting may be made detectable; we can minimize increases to the surface of passive fingerprinting; and cookie-like mechanisms can be made clearable.

Some implementers and some users may be willing to accept reduced functionality or decreased performance in order to minimize browser fingerprinting. Documenting which features have fingerprinting risk eases the work of implementers building modes for these at-risk users; minimizing fingerprinting even in cases where common implementations will have easy active fingerprintability allows such users to reduce the functionality trade-offs necessary. Making browser fingerprinting more detectable also contributes to mitigations outside the standardization process; for example, though regulatory or policy means [TAG-UNSANCTIONED].

5. Identifying fingerprinting surface and evaluating severity

ISSUE 1

Re: Issue 13: actionability through a decision tree or other, this section is intended to provide advice on identifying fingerprinting surface and evaluating severity based on several factors.

To mitigate browser fingerprinting in your specification, first identify what features could be used for browser fingerprinting. The ***fingerprinting surface*** of a user agent is the set of observable characteristics that can be used in concert to identify a user, user agent or device or correlate its activity.

Data sources that may be used for browser fingerprinting include:

- user configuration
- device characteristics
- environmental characteristics *(e.g. sensor readings)*
- operating system characteristics
- user behavior
- browser characteristics

The Tor Browser design document has more details on these sources and their relative priorities; this document adds environmental characteristics in that sensor readings or data access may distinguish a user, user agent or device by information about the environment (location, for example).

Second, for each identified feature, consider the severity for the privacy impacts described above (1.2 Privacy impacts and threat models) based on the following factors:

entropy
: How distinguishing is this new surface? Consider both the possible variations and the likely distribution of values. Adding 1-bit of entropy is typically of less concern; 30-some bits of entropy would be enough to uniquely identify every individual person.

detectability
: Will use of this feature for browser fingerprinting be observable to the user agent or likely to be discoverable by researchers? Because detectability is an important — and perhaps the most feasible — mitigation, increases to the surface for passive fingerprinting are of particular concern and should be avoided.

persistence
: How long will the characteristics of this fingerprinting surface stay unchanged? Can users control or re-set these values to prevent long-lived identification? While short-lived characteristics may still enable unexpected correlation of activity (for example, between two browser profiles on the same device), persistent or permanent identifiers are particularly concerning for the lack of user control.

availability
: Will this surface be accessible to the "drive-by Web" or only in certain contexts where a user has granted a particular sensor permission or directly authenticated? While browser fingerprinting is still something to mitigate in the permissioned context, the concern that a feature will end up used primarily for fingerprinting is reduced.

scope
: Is this surface consistent across origins or only within a single origin? In general, characteristics or identifiers that are tied to a particular origin are of less concern and can be handled with the same tools as HTTP cookies.

While we do not recommend specific trade-offs, these factors can be used to weigh increases to that surface (6.1 Weighing increased fingerprinting surface) and suggest appropriate mitigations. Although each factor may suggest specific mitigations, in weighing whether to add fingerprinting surface they should be considered in concert. For example, access to a new set of characteristics about the user may be high entropy, but be of less concern because it has limited availability and is easily detectable. A cross-origin, drive-by-accessible, permanent, passive unique identifier is incompatible with our expectations for privacy on the Web.

In conducting this analysis, it may be tempting to dismiss certain fingerprinting surface in a specification because of a comparison to fingerprinting surface exposed by other parts of the Web platform or other layers of the stack. Be cautious about making such claims. First, while similar information may be available through other means, similar is not identical: information disclosures may not be exactly the same and fingerprintability is promoted by combining these distinct sources. Second, where identical entropy is present, other factors of severity or accessibility may differ and those factors are important for feasible mitigation. Third, the platform is neither monolithic nor static; not all other features are implemented in all cases and may change (or be removed) in the future. Fourth, circular dependencies are a danger when so many new features are under development; two specifications sometimes refer to one another in arguing that fingerprinting surface already exists. It is more useful to reviewers and implementers to consider the fingerprinting surface provided by the particular Web feature itself, with specific references where surface may be accessible through other features as well.

6. Mitigations

6.1 Weighing increased fingerprinting surface

Web specification authors regularly attempt to strike a balance between new functionality and fingerprinting surface. For example, feature detection functionality allows for progressive enhancement with a small addition to fingerprinting surface; detailed enumerations of plugins, fonts, connected devices may provide a large fingerprinting surface with minimal functional support.

Authors and Working Groups determine the appropriate balance between these properties on a case-by-case basis, given their understanding of the functionality, its likely implementations and the entropy of increased fingerprinting surface. However, given the distinct privacy impacts described above and in order to improve consistency across specifications, these practices provide some guidance:

Best Practice 1: Avoid unnecessary or severe increases to fingerprinting surface.

Consider each of the severity factors described above and whether comparable functionality is feasible without adding to the fingerprinting surface. In particular, unless a feature cannot reasonably be designed in any other way, increased passive fingerprintability should be avoided. Passive fingerprinting allows for easier identification, without opportunities for external detection or control by users or third parties.

Best Practice 2: Mark features that contribute to fingerprintability.

Where a feature does contribute to the fingerprinting surface, indicate that impact, by explaining the effect (and any known implementer mitigations) and marking the relevant section with a fingerprinting icon, as this paragraph is.

6.2 Standardization

Specifications can mitigate against fingerprintability through standardization; by defining a consistent behavior, conformant implementations won't have variations that can be used for browser fingerprinting.

Randomization of certain browser characteristics has been proposed as a way to combat browser fingerprinting. While this strategy may be pursued by some implementations, we expect in general it will be more effective for us to standardize or null values rather than setting a range over which they can vary. The Tor Browser design provides more detailed information, but in short: it's difficult to measure how well randomization will work as a mitigation and it can be costly to implement in terms of usability (varying functionality or design in unwanted ways), processing (generating random numbers) and development (including the cost of introducing new security vulnerabilities).

> ***Best Practice 3: Specify orderings and non-functional differences.***
>
> To reduce unnecessary entropy, specify aspects of API return values and behavior that don't contribute to functional differences. For example, if the ordering of return values in a list has no semantic value, specify a particular ordering (alphabetical order by a defined algorithm, for example) so that incidental differences don't expose fingerprinting surface.
>
> Access to a list of system fonts via Flash or Java plugins notably returns the list sorted not in a standard alphabetical order, but in an unspecified order specific to the system. This ordering adds to the entropy available from that plugin in a way that provides no functional advantage. (See Collecting System Fonts via Flash Plugins.)

Standardization does *not* need to attempt to hide all differences between different browsers (e.g. Edge and Chrome); implemented functionality and behavior differences will always exist between different implementations. For that reason, removing `User-Agent` headers altogether is not a goal. However, variation in the `User-Agent` string that reveals additional information about the user or device has been shown to provide substantial fingerprinting surface (see "Beauty and the Beast" in A. Research, below).

6.3 Detectability

Where a client-side API provides some fingerprinting surface, authors can still mitigate the privacy concerns via detectability. If client-side fingerprinting activity is to some extent distinguishable from functional use of APIs, user agent implementations may have an opportunity to prevent ongoing fingerprinting or make it observable to users and external researchers (including academics or relevant regulators) who may be able to detect and investigate the use of fingerprinting.

Best Practice 4: Design APIs to access only the entropy necessary.

Following the basic principle of data minimization [RFC6973], design your APIs such that a site can access (and does access by default) only the entropy necessary for particular functionality.

Authors might design an API to allow for querying of a particular value, rather than returning an enumeration of all values. User agents and researchers can then more easily distinguish between sites that query for one or two particular values (gaining minimal entropy) and those that query for all values (more likely attempting to fingerprint the browser); or implementations can cap the number of different values. For example, Tor Browser limits the number of fonts that can be queried with a `browser.display.max_font_attempts` preference.

The granularity or precision of information returned can be minimized in order to reduce entropy. For example, implementations of the Battery Status API [BATTERY-STATUS] allowed for high precision (double-precision, or 15-17 significant digits) readings of the current battery level, which provided a short-term identifier that could be used to correlate traffic across origins or clearance of local state. Rounding off values to lower precision mitigates browser fingerprinting while maintaining functional use cases. Alternatively, providing Boolean or a small enumeration of values might provide functionality without revealing underlying details; for example, the Boolean `near` property in the Proximity Sensor API [PROXIMITY].

For more information, see:

- Data Minimization in Web APIs, Draft TAG Finding, September 2011
- Device API Privacy Requirements, DAP Working Group Note, June 2010
- Generic Sensor API: Security and privacy considerations, Editor's Draft, May 2016.
- The leaking battery: A privacy analysis of the HTML5 Battery Status API, February 2016.

Implementers can facilitate detectability by providing or enabling instrumentation so that users or third parties are able to calculate when fingerprinting surface is being accessed. Of particular importance for instrumentation are: access to all the different sources of fingerprinting surface; identification of the originating script; avoiding exposure that instrumentation is taking place. Beyond the minimization practice described above, these are largely implementation-specific (rather than Web specification) features.

If your specification exposes some fingerprinting surface (whether it's active or passive), some implementers (e.g. the Tor Browser) are going to be compelled to disable those features for certain privacy-conscious users.

Best Practice 5: Enable graceful degradation for privacy-conscious users or implementers.

Following the principle of progressive enhancement, and to avoid further divergence (which might itself expose variation in users), consider whether some functionality in your specification is still possible if fingerprinting surface features are disabled.

Explicit hooks or API flags may be used so that browser extensions or certain user agents can easily disable specific features. For example, the origin-clean flag allows control over whether an image canvas can be read, a significant fingerprinting surface.

6.4 Clearing all local state

Features which enable storage of data on the client and functionality for client- or server-side querying of that data can increase the ease of cookie-like fingerprinting. Storage can vary between large amounts of data (for example, the Web Storage API) or just a binary flag (has or has not provided a certain permission; has or has not cached a single resource).

Best Practice 6: Avoid unnecessary new local state mechanisms.

If functionality does not require maintaining client-side state in a way that is subsequently queryable (or otherwise observable), avoid creating a new cookie-like feature. Can the functionality be accomplished with existing HTTP cookies or an existing JavaScript local storage API?

For example, the Flash plugin's Local Shared Objects (LSOs) have often been used to duplicate and re-spawn HTTP cookies cleared by the user. (See "Flash Cookies and Privacy", Soltani et al., August 2009.)

Where features do require setting and retrieving local state, there are ways to mitigate the privacy impacts related to unexpected cookie-like behavior; in particular, you can help implementers prevent "permanent", "zombie", "super" or "evercookies".

Best Practice 7: Highlight any local state mechanisms to enable simultaneous clearing.

Clearly note where state is being maintained and could be queried and provide guidance to implementers on enabling simultaneous deletion of local state for users. Such functionality can mitigate the threat of "evercookies" because the presence of state in one such storage mechanism can't be used to persist and re-create an identifier.

Permanent or persistent data (including any identifiers) are of particular risk because they undermine the ability for a user to clear or re-set the state of their device or to maintain different identities.

Best Practice 8: Limit permanent or persistent state.

Permanent identifiers or other state (for example, identifiers or keys set in hardware) should typically not be exposed. Where necessary, access to such identifiers would require user permission (however, explaining the implications of such permission to users may be difficult) and limitation to a particular origin (however, server-side collusion between origins will be difficult to detect). As a result, your design should not rely on saving and later querying data on the client beyond a user's clearing cookies or other local state. That is, you should not expect any local state information to be permanent or to persist longer than other local state.

Though not strictly browser fingerprinting, there are other privacy concerns regarding user tracking for features that provide local storage of data. Mitigations suggested in the Web Storage API specification include: white-listing, black-listing, expiration and secure deletion [WEBSTORAGE-user-tracking].

6.5 Do Not Track

Expressions of, and compliance with, a Do Not Track signal does not inhibit the capability of browser fingerprinting, but may mitigate some user concerns about fingerprinting, specifically around tracking as defined in those specifications [TRACKING-DNT] [TRACKING-COMPLIANCE] and as implemented by services that comply with those user preferences. That is, DNT can mitigate concerns with cooperative sites.

The use of DNT in this way typically does not require changes to other functional specifications. If your specification expects a particular behavior upon receiving a particular DNT signal, indicate that with a reference to [TRACKING-DNT]. If your specification introduces a new communication channel that could be used for tracking, you might wish to define how a DNT signal should be communicated.

A. Research

Some browser developers maintain pages on browser fingerprinting, including: potential mitigations or modifications necessary to decrease the surface of that browser engine; different vectors that can be used for fingerprinting; potential future work. These are not cheery, optimistic documents.

- "Technical analysis of client identification mechanisms". The Chromium Projects.
- WebKit Wiki: Fingerprinting
- Mozilla Wiki: Fingerprinting
- The Design and Implementation of the Tor Browser: Cross-Origin Fingerprinting Unlinkability

What are the key papers to read here, historically or to give the latest on fingerprinting techniques? What are some areas of open research that might be relevant?

- Eckersley, Peter. "How unique is your web browser?" *Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 2010.
- Mowery, Keaton, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. "Fingerprinting Information in JavaScript Implementations." In *Web 2.0 Security and Privacy*, 2011.
- Yen, Ting-Fang, et al. "Host fingerprinting and tracking on the web: Privacy and security implications." *Proceedings of NDSS*. 2012.
- Mowery, Keaton, and Hovav Shacham. "Pixel perfect: Fingerprinting canvas in HTML5." *Web 2.0 Security and Privacy*, 2012.
- Mattioli, Dana. "On Orbitz, Mac Users Steered to Pricier Hotels". *Wall Street Journal*, August 23, 2012.
- Gunes Acar et al. "FPDetective: dusting the web for fingerprinters." In *CCS '13*.
- Nikiforakis, Nick, et al. "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting." *IEEE Symposium on Security and Privacy (S&P 2013)*, 2013.
- G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, C. Diaz. "The Web never forgets: Persistent tracking mechanisms in the wild." In *Proceedings of CCS 2014*, Nov. 2014.
- Steven Englehardt, Arvind Narayanan. "Online tracking: A 1-million-site measurement and analysis." May 2016.

- Pierre Laperdrix, Walter Rudametkin, Benoit Baudry. "Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints." *IEEE Symposium on Security and Privacy (S&P 2016)*, May 2016.

Testing

A non-exhaustive list of sites that allow the visitor to test their configuration for fingerprintability.

- amiunique.org (INRIA)
- panopticlick.eff.org (EFF)
- BrowserSPY.dk
- pet-portal cross-browser fingerprinting test
- p0f v3 (purely passive fingerprinting)

B. Acknowledgements

Many thanks to Robin Berjon for ReSpec and to Tobie Langel for Github advice; to the Privacy Interest Group and the Technical Architecture Group for review; to the Tor Browser designers for references and recommendations; and to Christine Runnegar for contributions.

C. References

C.1 Informative references

[BATTERY-STATUS]
Battery Status API. Anssi Kostiainen; Mounir Lamouri. W3C. 7 July 2016. W3C Candidate Recommendation. URL: https://www.w3.org/TR/battery-status/

[EVERCOOKIE]
evercookie - virtually irrevocable persistent cookies. Samy Kamkar.September 2010. URL: http://samy.pl/evercookie/

[NDSS-FINGERPRINTING]
Host Fingerprinting and Tracking on the Web: Privacy and Security Implications. Ting-Fang Yen; Yinglian Xie; Fang Yu; Roger Peng Yu; Martin Abadi. In Proceedings of the Network and Distributed System Security Symposium (NDSS). February 2012. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=156901

[PROXIMITY]
Proximity Sensor. Anssi Kostiainen; Rijubrata Bhaumik. W3C. 19 July 2016. W3C Working Draft. URL: https://www.w3.org/TR/proximity/

[RFC6265]
HTTP State Management Mechanism. A. Barth. IETF. April 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6265

[RFC6454]
The Web Origin Concept. A. Barth. IETF. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6454

[RFC6973]
Privacy Considerations for Internet Protocols. A. Cooper; H. Tschofenig; B. Aboba; J. Peterson; J. Morris; M. Hansen; R. Smith. IETF. July 2013. RFC. URL: http://www.rfc-editor.org/rfc/rfc6973.txt

[TAG-UNSANCTIONED]
Unsanctioned Web Tracking. Mark Nottingham. W3C Technical Architecture Group. 17 July 2015. URL: https://w3ctag.github.io/unsanctioned-tracking/

[TRACKING-COMPLIANCE]
Tracking Compliance and Scope. Nick Doty; Justin Brookman; Heather West; Sean Harvey; Erica Newland. W3C. July 2015. W3C Last Call Working Draft. URL: http://www.w3.org/TR/tracking-compliance/

[TRACKING-DNT]
Tracking Preference Expression (DNT). Roy Fielding; David Singer. W3C. April 2014. W3C Last Call Working Draft. URL: http://www.w3.org/TR/tracking-dnt/

[WEBSTORAGE-user-tracking]
Web Storage > Privacy > User tracking. Ian Hickson. W3C. July 2013. Rec. URL: http://www.w3.org/TR/2013/REC-webstorage-20130730/#user-tracking

↑