4) Tracking

20.10.21

- Tracking is ubiquitous and has led to "surveillance capitalism" of today

	2020 profit/USD	no. user	profit/ user
Google	40 E9	3 E9	13
Facebook	29 E9	2.8 E9	10
Amazon	21 E9	300 E6	70*
Apple	57 E9	1.5 E9	38*
Microsoft	44 E9	1.5 E9	29*

* sell stuff / hardware / software

4.1) Prelude – Intel's Pentium serial number of 1999

- Intel's CPU included serial number

and instruction to read it out

- Big debate followed, focused on 'privacy'
- Intel gave in, made this instruction optional
- Two interesting developments

1) Intel invested into privacy via advanced crypto. / ZKP

- TPM: Direct Anonymous Attestation (DAA)
  ~2005-10
- Enhanced Privacy ID (EPID)
  ~2010-

- Both contain ZKP and anonymous credentials (IBM ZRL)

⇒ None is really used

2) Today there are many different unique identifiers in OS or programs,

- Windows GUID
- Apple ID
- Goole ID
- GMail → Google

⇒ Nobody seems to care so much

4.2) Active tracking methods

1) Cookies

- Web browers
- HTTP is stateless
  ⇒ Need method to provide continuity and sessions

- Cookies address this
  => personalization

  => logins, shopping cart

  => tracking

- Servers set cookies in a browser
- Browser sends cookies in every subsequent request to original server

1st-party cookie

- Set by the server in URL / from which DOM originates
  - Needed for state continuity

3-party cookie

- When DOM includes some content from other servers, client sends request there,

and servers may respond with 3-party cookies

→ primarily used for tracking

Google: analytics, code, CSS..
Facebook & co: "share" and "like"

From 2022 on, Google Chrome will block 3rd party cookies by default.

For a list of most prominent trackers, see [EN16].

2) Tracking pixels

Typically 1x1 transparent images included from external server, via the DOM.

- Used on websites
- Used in emails - viewing receipt

3) Identifiers on device or browser

- Mobile:
  - IMSI ....
  - GAID – Google advertising ID
  - IDFA – ID for advertisers
    ↑ opt-in since 2020
- Desktop: not very common, but recall Intel DAA/EPID

4) Evercookie [AEE3ND 14, §2, §4]

Using other methods for storage in browsers, goal is to recreate same cookie on every visit.

Using:
- Flash (historic, deprecated)
- localStorage, sessionStorage ...
- ETags (HTTP element to facilitate caching)

...

5) Cookie syncing [AEEJND14, §5]

Correlate 1st-party cookies sent by different servers, across domains.

- Domain A places cookie and makes browser include some content from Domain B via personalized URL, which includes tracking data in URL

- Domain B can associate this with identifier used by A

- All domains merge all data obtained individually.

4.3) Passive tracking

- Client takes no action
- Trackers are not visible on clients
- Usually called "fingerprinting"

6) Browser-fingerprinting (primarily desktop)

- Browsers have many identifying features sent to servers

[LBBA20]

Table 1. Example of a Browser Fingerprint

Attribute	Source	Example
User agent	HTTP header	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.119 Safari/537.36
Accept	HTTP header	text/html,application/xhtml+xml,application/xml;q=0.9, image/webp,image/apng,*/*;q=0.8
Content encoding	HTTP header	gzip, deflate, br
Content language	HTTP header	en-US,en;q=0.9
List of plugins	JavaScript	Plugin 1: Chrome PDF Plugin. Plugin 2: Chrome PDF Viewer. Plugin 3: Native Client. Plugin 4: Shockwave Flash...
Cookies enabled	JavaScript	yes
Use of local/session storage	JavaScript	yes
Timezone	JavaScript	−60 (UTC+1)
Screen resolution and color depth	JavaScript	1920 × 1200 × 24
List of fonts	Flash or JS	Abyssinica SIL,Aharoni CLM,AR PL UMing CN,AR PL UMing HK,AR PL UMing TW...
List of HTTP headers	HTTP headers	Referer X-Forwarded-For Connection Accept Cookie Accept-Language Accept-Encoding User-Agent Host
Platform	JavaScript	Linux x86_64
Do Not Track	JavaScript	yes
Canvas	JavaScript	Cwm fjordbank glyphs vext quiz, 😃 Cwm fjordbank glyphs vext quiz, 😃
WebGL Vendor	JavaScript	NVIDIA Corporation
WebGL Renderer	JavaScript	GeForce GTX 650 Ti/PCIe/SSE2
Use of an ad blocker	JavaScript	yes

[HMKHG15]

Table 3: Feature Data Types and Example Values

Feature	Type	Example
devicefingerprint	string	4812169833755445458
revisitor	bit	1
ismobile	bit	1
cookie_id	string	QoSQIymCwjg0augzs D41-1415043670.767
localstorage_id	string	rQG4fVJaDBNFtOyKd CL1-1415011415.67
mimetypes	text	[{"n":"video/3gpp2","d":"3GPP2 media","f":"3g2,3gp2"}, ...]
mimetype_hash	hash	b96eebf2fd3fff0e165d77e75474ffaf
plugins	text	[{"n":"Shockwave Flash","d": "Shockwave Flash 11.1 r115",...}]
plugins_hash	hash	4e23a836cea77cf4af09affff2b64a75
plugins_num	int	6
canvas_hash	hash	ea907f4cd06cf0f310d7acf62f2ffff6
useragent	text	Mozilla/5.0 (...)
vendor	text	Google Inc.
productsub	text	20030107
is_chrome	bit	0
popup_blocker	bit	1
navigatorlanguage	text	en-en
filesystem_access	bit	1
cookies_enabled	bit	1
dnt_enabled	bit	0
java_enabled	bit	0
loginstatus	bitstring	10111
history	bitstring	1000100000
screen_height	int	960
screen_width	int	600
display_colordepth	int	32
display_orientation	text	landscape
platform	text	Linux armv7l
operation_system	text	iOS 7.1
is_Android	bit	0
is_iOS	bit	1
touchpoints	int	5
has_vibration	bit	1
airplay_ref	bit	1
typingspeed	int	229
accelerometer key	float	938.143359751
connection	text	wifi
hostname	text	ip-xx-xx-xx-xxx.web.provider.com
hostname_wildcard	text	*.web.provider.com
timezone_id	int	662525310
ipaddress	string	xx.xxx.xx.xxx
country	text	Germany
city	text	Bochum

* Canvas fingerprinting

Fonts, display features ... influence how rendered on a device

Measure this!

[AEEJND 14]

Fingerprinting script	Number of including sites	Text drawn into the canvas
ct1.addthis.com/static/r07/core130.js	5282	Cwm fjordbank glyphs vext quiz,😃
i.ligatus.com/script/fingerprint.min.js	115	http://valve.github.io
src.kitcode.net/fp2.js	68	http://valve.github.io
admicro1.vcmedia.vn/fingerprint/figp.js	31	http://admicro.vn/
amazonaws.com/af-bdaz/bquery.js	26	Centillion
*.shorte.st/js/packed/smeadvert-intermediate-ad.js	14	http://valve.github.io
stat.ringier.cz/js/fingerprint.min.js	4	http://valve.github.io
cya2.net/js/STAT/89946.js	3	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz0123456789+/
images.revtrax.com/RevTrax/js/fp/fp.min.jsp	3	http://valve.github.io
pof.com	2	http://www.plentyoffish.com
*.rackcdn.com/mongoose.fp.js	2	http://api.gonorthleads.com
9 others*	9	*(Various)*
TOTAL	5559 *(5542 unique[1])*	-

Table 1: Canvas fingerprinting domains found on Top Alexa 100K sites.
*: Some URLs are truncated or omitted for brevity. See Appendix for the complete list of URLs.
1: Some sites include canvas fingerprinting scripts from more than one domain.

fingerprinting scripts than the ones within the 1,000-10,000 range.

Note that the URL *http://valve.github.io*, printed by many scripts onto the canvas, belongs to the developer of an open source fingerprinting library[7]. Furthermore, all scripts except one use the same colors for the text and background shape. This similarity is possibly due to the use of the publicly available open source fingerprinting library *fingerprintjs* [51]. Figure 4 shows five different canvas images used by different canvas fingerprinting scripts. The images are generated by intercepting the canvas pixel data extracted by the scripts listed in Table 1.

Figure 4: Different images printed to canvas by fingerprinting scripts. Note that the phrase *"Cwm fjordbank glyphs vext quiz"* in the top image is a *perfect pangram*, that is, it contains all the letters of the English alphabet only once to maximize diversity of the outcomes with the shortest possible string.

and adds new tests to extract more entropy from the canvas image. Specifically, we found that in addition to the techniques outlined in Mowery and Shacham's canvas fingerprinting paper [32] AddThis scripts perform the following tests:

- Drawing the text twice with different colors and the default fallback font by using a fake font name, starting with *"no-real-font-"*.
- Using the perfect pangram[8] *"Cwm fjordbank glyphs vext quiz"* as the text string
- Checking support for drawing Unicode by printing the character *U+1F603 a smiling face with an open mouth.*
- Checking for canvas *globalCompositeOperation* support.
- Drawing two rectangles and checking if a specific point is in the path by the *isPointInPath* method.

By requesting a non-existent font, the first test tries to employ the browser's default fallback font. This may be used to distinguish between different browsers and operating systems. Using a perfect pangram, which includes a single instance of each letter of the English alphabet, the script enumerates all the possible letter forms using the shortest string. The last three tests may be trying to uncover the browser's support for the canvas features that are not equally supported. For instance, we found that the Opera browser cannot draw the requested Unicode character, *U+1F603.*

* Font list

* Device audio feature

  - Send audio signal, have it processed, measure it!

* WebRTC

  WebRTC tech. provides streaming features for browsers

  - reveal local IP behind NAT
  - API characteristics

* Battery API [OEN17]

  - Existed from 2012 - 2016
  - Reveals a lot of info
  - Studies showed how users acted differently based this sensitive info
  - W3C decided to remove

7) Device fingerprinting (mostly mobile)

- Mobile differs from desktop
- In pos. and neg. ways

Makes tracking easier	Makes tracking harder
Users have no direct control	Users have no direct control
• fewer options to user	• fewer unique features
• default browsers offer less customization	• devices are homogeneous
• many identifiers available	• vendors (slowly) allow hiding of identifiers
• some Android browsers send OS version and	

phone model in user-agent

Rest of doc contains website includes, as shown during lecture.

(Last two commercial fingerprinting toolkit vendors omitted, to limit length. They were:

- ipqualityscore.com/mobile-device-fingerprinting-sdk
- blog.getsocial.im/device-fingerprinting-for-mobile-attribution

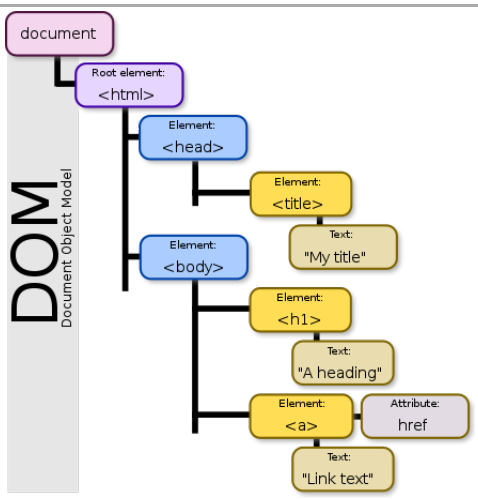
)

Document Object Model

The **Document Object Model** (**DOM**) is a cross-platform and language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document. Nodes can have event handlers attached to them. Once an event is triggered, the event handlers get executed.[2]

The principal standardization of the DOM was handled by the World Wide Web Consortium (W3C), which last developed a recommendation in 2004. WHATWG took over the development of the standard, publishing it as a living document. The W3C now publishes stable snapshots of the WHATWG standard.

Document Object Model



Example of DOM hierarchy in an HTML document

First published	October 1, 1998
Latest version	DOM4[1] November 19, 2015
Organization	World Wide Web Consortium, WHATWG
Base standards	WHATWG DOM Living Standard (https://dom.spec.whatwg.org/) W3C DOM4 (https://www.w3.org/TR/dom/)
Abbreviation	DOM

Contents

History

The history of the Document Object Model is intertwined with the history of the "browser wars" of the late 1990s between Netscape Navigator and Microsoft Internet Explorer, as well as with that of JavaScript and JScript, the first scripting languages to be widely implemented in the JavaScript engines of web browsers.

JavaScript was released by Netscape Communications in 1995 within Netscape Navigator 2.0. Netscape's competitor, Microsoft, released Internet Explorer 3.0 the following year with a reimplementation of JavaScript called JScript. JavaScript and JScript let web developers create web pages with client-side interactivity. The limited facilities for detecting user-generated events and modifying the HTML document in the first generation of these languages eventually became known as "DOM Level 0" or "Legacy DOM." No independent standard was developed for DOM Level 0, but it was partly described in the specifications for HTML 4.

Legacy DOM was limited in the kinds of elements that could be accessed. Form, link and image elements could be referenced with a hierarchical name that began with the root document object. A hierarchical name could make use of either the names or the sequential index of the traversed elements. For example, a form input element could be accessed as either `document.formName.inputName` or `document.forms[0].elements[0]`.

The Legacy DOM enabled client-side form validation and simple interface interactivity like creating tooltips.

In 1997, Netscape and Microsoft released version 4.0 of Netscape Navigator and Internet Explorer respectively, adding support for Dynamic HTML (DHTML) functionality enabling changes to a loaded HTML document. DHTML required extensions to the rudimentary document object that was available in the Legacy DOM implementations. Although the Legacy DOM implementations were largely compatible since JScript was based on JavaScript, the DHTML DOM extensions were developed in parallel by each browser maker and remained incompatible. These versions of the DOM became known as the "Intermediate DOM."

After the standardization of ECMAScript, the W3C DOM Working Group began drafting a standard DOM specification. The completed specification, known as "DOM Level 1", became a W3C Recommendation in late 1998. By 2005, large parts of W3C DOM were well-supported by common ECMAScript-enabled browsers, including Microsoft Internet Explorer version 6 (from 2001), Opera, Safari and Gecko-based browsers (like Mozilla, Firefox, SeaMonkey and Camino).

Standards

The W3C DOM Working Group published its final recommendation and subsequently disbanded in 2004. Development efforts migrated to the WHATWG, which continues to maintain a living standard.[3] In 2009, the Web Applications group reorganized DOM activities at the W3C.[4] In 2013, due to a lack of progress and the impending release of HTML5, the DOM Level 4 specification was reassigned to the HTML Working Group to expedite its completion.[5] Meanwhile, in 2015, the Web Applications group was disbanded and DOM stewardship passed to the Web Platform group.[6] Beginning with the publication of DOM Level 4 in 2015, the W3C creates new recommendations based on snapshots of the WHATWG standard.

- DOM Level 1 provided a complete model for an entire HTML or XML document, including the means to change any portion of the document.

- DOM Level 2 was published in late 2000. It introduced the `getElementById` function as well as an event model and support for XML namespaces and CSS.
- DOM Level 3, published in April 2004, added support for XPath and keyboard event handling, as well as an interface for serializing documents as XML.
- DOM Level 4 was published in 2015. It is a snapshot of the WHATWG living standard.[7]

Applications

Web browsers

To render a document such as a HTML page, most web browsers use an internal model similar to the **DOM**. The nodes of every document are organized in a tree structure, called the *DOM tree*, with the topmost node named as "Document object". When an HTML page is rendered in browsers, the browser downloads the HTML into local memory and automatically parses it to display the page on screen. However, the DOM does not necessarily need to be represented as a tree,[8] and some browsers have used other internal models.[9]

JavaScript

When a web page is loaded, the browser creates a Document Object Model of the page, which is an object oriented representation of an HTML document that acts as an interface between JavaScript and the document itself. This allows the creation of dynamic web pages,[10] because within a page JavaScript can:

WHATWG DOM

- add, change, and remove any of the HTML elements and attributes
- change any of the CSS styles
- react to all the existing events
- create new events

Implementations

Because the DOM supports navigation in any direction (e.g., parent and previous sibling) and allows for arbitrary modifications, an implementation must at least buffer the document that has been read so far (or some parsed form of it).[11]

Layout engines

MDN Web Docs
moz://a

Using HTTP cookies

An **HTTP cookie** (web cookie, browser cookie) is a small piece of data that a server sends to a user's web browser. The browser may store the cookie and send it back to the same server with later requests. Typically, an HTTP cookie is used to tell if two requests come from the same browser—keeping a user logged in, for example. It remembers stateful information for the stateless HTTP protocol.

Cookies are mainly used for three purposes:

Session management

Logins, shopping carts, game scores, or anything else the server should remember

Personalization

User preferences, themes, and other settings

Tracking

Recording and analyzing user behavior

Cookies were once used for general client-side storage. While this made sense when they were the only way to store data on the client, modern storage APIs are now recommended.

Cookies are sent with every request, so they can worsen performance (especially for mobile data connections). Modern APIs for client storage are the Web Storage API (`localStorage` and `sessionStorage`) and IndexedDB.

> **Note:** To see stored cookies (and other storage that a web page can use), you can enable the Storage Inspector in Developer Tools and select Cookies from the storage tree.

Creating cookies

After receiving an HTTP request, a server can send one or more `Set-Cookie` headers with the response. The browser usually stores the cookie and sends it with requests made to the same server inside a `Cookie` HTTP header. You can specify an expiration date or time period after which the cookie shouldn't be sent. You can also set additional restrictions to a specific domain and path to limit where the cookie is sent. For details about the header attributes mentioned below, refer to the `Set-Cookie` reference article.

The `Set-Cookie` and `Cookie` headers

The `Set-Cookie` HTTP response header sends cookies from the server to the user agent. A simple cookie is set like this:

```
Set-Cookie: <cookie-name>=<cookie-value>
```

This instructs the server sending headers to tell the client to store a pair of cookies:

```
HTTP/2.0 200 OK
Content-Type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry

[page content]
```

Then, with every subsequent request to the server, the browser sends all previously stored cookies back to the server using the `Cookie` header.

```
GET /sample_page.html HTTP/2.0
Host: www.example.org
Cookie: yummy_cookie=choco; tasty_cookie=strawberry
```

> **Note:** Here's how to use the `Set-Cookie` header in various server-side applications:
>
> - PHP

- Node.JS
- Python
- Ruby on Rails

Define the lifetime of a cookie

The lifetime of a cookie can be defined in two ways:

- *Session* cookies are deleted when the current session ends. The browser defines when the "current session" ends, and some browsers use *session restoring* when restarting. This can cause session cookies to last indefinitely.
- *Permanent* cookies are deleted at a date specified by the `Expires` attribute, or after a period of time specified by the `Max-Age` attribute.

For example:

```
Set-Cookie: id=a3fWa; Expires=Thu, 31 Oct 2021 07:28:00 GMT;
```

> **Note:** When you set an `Expires` date and time, they're relative to the client the cookie is being set on, not the server.

If your site authenticates users, it should regenerate and resend session cookies, even ones that already exist, whenever a user authenticates. This approach helps prevent session fixation attacks, where a third party can reuse a user's session.

Restrict access to cookies

You can ensure that cookies are sent securely and aren't accessed by unintended parties or scripts in one of two ways: with the `Secure` attribute and the `HttpOnly` attribute.

A cookie with the `Secure` attribute is only sent to the server with an encrypted request over the HTTPS protocol. It's never sent with unsecured HTTP (except on localhost), which means

attackers man-in-the-middle can't access it easily. Insecure sites (with `http:` in the URL) can't set cookies with the `Secure` attribute. However, don't assume that `Secure` prevents all access to sensitive information in cookies. For example, someone with access to the client's hard disk (or JavaScript if the `HttpOnly` attribute isn't set) can read and modify the information.

A cookie with the `HttpOnly` attribute is inaccessible to the JavaScript `Document.cookie` API; it's only sent to the server. For example, cookies that persist in server-side sessions don't need to be available to JavaScript and should have the `HttpOnly` attribute. This precaution helps mitigate cross-site scripting (XSS) attacks.

Here's an example:

```
Set-Cookie: id=a3fWa; Expires=Thu, 21 Oct 2021 07:28:00 GMT; Secure; HttpOnly
```

Define where cookies are sent

The `Domain` and `Path` attributes define the *scope* of a cookie: what URLs the cookies should be sent to.

Domain attribute

The `Domain` attribute specifies which hosts can receive a cookie. If unspecified, the attribute defaults to the same host that set the cookie, *excluding subdomains*. If `Domain` *is* specified, then subdomains are always included. Therefore, specifying `Domain` is less restrictive than omitting it. However, it can be helpful when subdomains need to share information about a user.

For example, if you set `Domain=mozilla.org`, cookies are available on subdomains like `developer.mozilla.org`.

Path attribute

The `Path` attribute indicates a URL path that must exist in the requested URL in order to send the `Cookie` header. The `%x2F` ("/") character is considered a directory separator, and subdirectories match as well.

subdirectories match as well.

For example, if you set `Path=/docs`, these request paths match:

- `/docs`
- `/docs/`
- `/docs/Web/`
- `/docs/Web/HTTP`

But these request paths don't:

- `/`
- `/docsets`
- `/fr/docs`

SameSite attribute

The `SameSite` attribute lets servers specify whether/when cookies are sent with cross-site requests (where Site is defined by the registrable domain). This provides some protection against cross-site request forgery attacks (CSRF). It takes three possible values: `Strict`, `Lax`, and `None`.

With `Strict`, the cookie is only sent to the site where it originated. `Lax` is similar, except that cookies are sent when the user *navigates* to the cookie's origin site. For example, by following a link from an external site. `None` specifies that cookies are sent on both originating and cross-site requests, but *only in secure contexts* (i.e., if `SameSite=None` then the `Secure` attribute must also be set). If no `SameSite` attribute is set, the cookie is treated as `Lax`.

Here's an example:

```
Set-Cookie: mykey=myvalue; SameSite=Strict
```

> **Note:** The standard related to `SameSite` recently changed (MDN documents the new

behavior above). See the cookies Browser compatibility table for information about how the attribute is handled in specific browser versions:

- `SameSite=Lax` is the new default if `SameSite` isn't specified. Previously, cookies were sent for all requests by default.
- Cookies with `SameSite=None` must now also specify the `Secure` attribute (they require a secure context).

Cookie prefixes

Because of the design of the cookie mechanism, a server can't confirm that a cookie was set from a secure origin or even tell *where* a cookie was originally set.

A vulnerable application on a subdomain can set a cookie with the `Domain` attribute, which gives access to that cookie on all other subdomains. This mechanism can be abused in a *session fixation* attack. See session fixation for primary mitigation methods.

As a defense-in-depth measure, however, you can use *cookie prefixes* to assert specific facts about the cookie. Two prefixes are available:

`__Host-`

If a cookie name has this prefix, it's accepted in a `Set-Cookie` header only if it's also marked with the `Secure` attribute, was sent from a secure origin, does *not* include a `Domain` attribute, and has the `Path` attribute set to `/`. This way, these cookies can be seen as "domain-locked".

`__Secure-`

If a cookie name has this prefix, it's accepted in a `Set-Cookie` header only if it's marked with the `Secure` attribute and was sent from a secure origin. This is weaker than the `__Host-` prefix.

The browser will reject cookies with these prefixes that don't comply with their restrictions. Note that this ensures that subdomain-created cookies with prefixes are either confined to the subdomain or ignored completely. As the application server only checks for a specific cookie

subdomain or ignored completely. As the application server only checks for a specific cookie name when determining if the user is authenticated or a CSRF token is correct, this effectively acts as a defense measure against session fixation.

> **Note:** On the application server, the web application *must* check for the full cookie name including the prefix. User agents *do not* strip the prefix from the cookie before sending it in a request's `Cookie` header.

For more information about cookie prefixes and the current state of browser support, see the Prefixes section of the Set-Cookie reference article.

JavaScript access using Document.cookie

You can create new cookies via JavaScript using the `Document.cookie` property. You can access existing cookies from JavaScript as well if the `HttpOnly` flag isn't set.

```
document.cookie = "yummy_cookie=choco";
document.cookie = "tasty_cookie=strawberry";
console.log(document.cookie);
// logs "yummy_cookie=choco; tasty_cookie=strawberry"
```

Cookies created via JavaScript can't include the `HttpOnly` flag.

Please note the security issues in the Security section below. Cookies available to JavaScript can be stolen through XSS.

Security

> **Note:** When you store information in cookies, keep in mind that all cookie values are visible to, and can be changed by, the end user. Depending on the application, you may want to use an opaque identifier that the server looks up, or investigate alternative authentication/confidentiality mechanisms such as JSON Web Tokens.

Ways to mitigate attacks involving cookies:

- Use the `HttpOnly` attribute to prevent access to cookie values via JavaScript.
- Cookies that are used for sensitive information (such as indicating authentication) should have a short lifetime, with the `SameSite` attribute set to `Strict` or `Lax`. (See SameSite attribute, above.) In browsers that support SameSite, this ensures that the authentication cookie isn't sent with cross-site requests. This would make the request effectively unauthenticated to the application server.

Tracking and privacy

Third-party cookies

A cookie is associated with a domain. If this domain is the same as the domain of the page you're on, the cookie is called a *first-party cookie*. If the domain is different, it's a *third-party cookie*. While the server hosting a web page sets first-party cookies, the page may contain images or other components stored on servers in other domains (for example, ad banners) that may set third-party cookies. These are mainly used for advertising and tracking across the web. For example, the types of cookies used by Google.

A third-party server can create a profile of a user's browsing history and habits based on cookies sent to it by the same browser when accessing multiple sites. Firefox, by default, blocks third-party cookies that are known to contain trackers. Third-party cookies (or just tracking cookies) may also be blocked by other browser settings or extensions. Cookie blocking can cause some third-party components (such as social media widgets) not to function as intended.

> **Note:** Servers can (and should) set the cookie SameSite attribute to specify whether or not cookies may be sent to third party sites.

Cookie-related regulations

Legislation or regulations that cover the use of cookies include:

- The General Data Privacy Regulation (GDPR) in the European Union
- The ePrivacy Directive in the EU
- The California Consumer Privacy Act

These regulations have global reach. They apply to any site on the *World Wide* Web that users from these jurisdictions access (the EU and California, with the caveat that California's law applies only to entities with gross revenue over 25 million USD, among things).

These regulations include requirements such as:

- Notifying users that your site uses cookies.
- Allowing users to opt out of receiving some or all cookies.
- Allowing users to use the bulk of your service without receiving cookies.

There may be other regulations that govern the use of cookies in your locality. The burden is on you to know and comply with these regulations. There are companies that offer "cookie banner" code that helps you comply with these regulations.

Other ways to store information in the browser

Another approach to storing data in the browser is the Web Storage API. The window.sessionStorage and window.localStorage properties correspond to session and permanent cookies in duration, but have larger storage limits than cookies, and are never sent to a server. More structured and larger amounts of data can be stored using the IndexedDB API, or a library built on it.

There are some techniques designed to recreate cookies after they're deleted. These are known as "zombie" cookies. These techniques violate the principles of user privacy and user control, may violate data privacy regulations, and could expose a website using them to legal liability.

Online tracking: A 1-million-site measurement and analysis is the largest and most detailed measurement of online tracking to date. We measure stateful (cookie-based) and stateless (fingerprinting-based) tracking, the effect of browser privacy tools, and "cookie syncing".

This measurement is made possible by our web measurement tool OpenWPM (https://github.com/citp/OpenWPM), a mature platform that enables fully automated web crawls using a full-fledged and instrumented browser.

Read the paper » (http://randomwalker.info/publications/OpenWPM_1_million_site_tracking_m

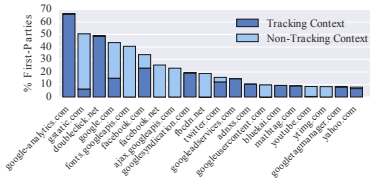
About

Authors: Steven Englehardt (http://senglehardt.com) and Arvind Narayanan (http://randomwalker.info) of Princeton University (*{ste,arvindn}@cs.princeton.edu*)

The study is part of the Princeton University's WebTAP project (https://webtap.princeton.edu/).

Tracking Results

The Long Tail of Online Tracking



During our January 2016 measurement of the top 1 million sites, our tool made over 90 million requests, assembling the largest dataset (to our knowledge) used for studying web tracking. With this scale we can answer many web tracking questions: *Who are the largest trackers? Which sites embed the largest number of trackers? Which tracking technologies are used, and who is using them?* and many more.

Findings

The total number of third parties present on at least two first parties is over 81,000, but the prevalence quickly drops off. Only 123 of these 81,000 are present on more than 1% of sites. This suggests that the number of third parties that a regular user will encounter on a daily basis is relatively small. The effect is accentuated when we consider that different third parties may be owned by the same entity. All of the top 5 third parties, as well as 12 of the top 20, are Google-owned domains. In fact, Google, Facebook, and Twitter are the only third-party entities present on more than 10% of sites.

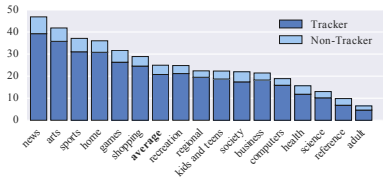
Third parties and HTTPS adoption

Mixed Content	Top 1M % FP	Top 50k % FP
Own	25.4%	29.6%
Favicon	1.4%	2.9%
Tracking	12.2%	27.0%
CDN	1.4%	2.9%
Non-tracking	43.4%	29.6%
Multiple causes	16.2%	8.1%

Third parties are a major roadblock to HTTPS adoption; insecure third-party resources loaded on secure sites (i.e. mixed content (https://developer.mozilla.org/en-US/docs/Security/Mixed_content) on HTTPS sites) will either be blocked or cause the browser to display security warnings. We find that a large number of third parties (54%) are only ever loaded over HTTP. A significant fraction of HTTP-default sites (26%) embed resources from at least one of the HTTP-only third parties on their homepage. These sites would be unable to upgrade to HTTPS without browsers displaying mixed content errors to their users, the majority of which (92%) would contain active content which would be blocked.

Around 78,000 first-party sites currently support HTTPS by default on their home pages. Nearly of these 8% load with mixed content warnings, of which 12% are caused by third-party trackers.

News sites have the most trackers



The level of tracking on different categories of websites varies considerably -- by almost an order of magnitude. The figure on the right shows average counts of tracking and non-tracking third parties per site for 100 of the top sites in each category.

Why is there so much variation? With the exception of the adult category, the sites on the low end of the spectrum are mostly sites which belong to government organizations, universities, and non-profit entities. This suggests that websites may be able to forgo advertising and tracking due to the presence of funding sources external to the web. Sites on the high end of the spectrum are largely those which provide editorial content. Since many of these sites provide articles for free, and lack an external funding source, they are pressured to monetize page views with significantly more advertising.

Does tracking protection work?

Users have two main ways to reduce their exposure to tracking: the browser's built in privacy features and extensions such as Ghostery or uBlock Origin. We used two test measurements of the top 55k sites with different blocking tools enabled: one with Ghostery enabled and set to block trackers, and one with Firefox's third-party cookie blocker enabled.

Findings

Firefox's third-party cookie blocking is very effective, only 237 sites (0.4%) have any third-party cookies set from a domain other than the landing page of the site. Most of these are for benign reasons, such as redirecting to the U.S. version of a non-U.S. site. We did find a handful of exceptions, including 32 that contained ID cookies. These sites appeared to be deliberately redirecting the landing page to a separate domain before redirecting back to the initial domain. Ghostery was effective at reducing both the number of third parties and ID cookies. The average number of third-party includes went down from 17.7 to 3.3, of which just 0.3 had third-party cookies (0.1 with IDs).

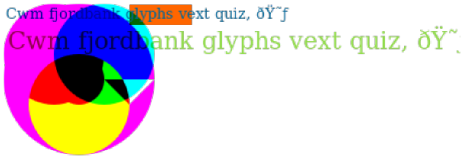
Fingerprinting Results

The growth (and diversity) of device fingerprinting.

Rank Interval	% of First-parties		
	Canvas	Canvas Font	WebRTC
[0,1K)	5.10%	2.50%	0.60%
[1K,10K)	3.91%	1.98%	0.42%
[10K,100K)	2.45%	0.86%	0.19%
[100K,1M)	1.31%	0.25%	0.06%

We examine four types of device fingerprinting. We provide updated Canvas fingerprinting measurements from our 2014 study (https://securehomes.esat.kuleuven.be/~gacar/persistent/). We also present findings on three techniques that have never been measured before: AudioContext fingerprinting, Canvas-Font fingerprinting, and WebRTC fingerprinting. The table on the right shows the percentage of sites on which each technique appears for different site ranks within the Alexa top 1 million.

Canvas Fingerprinting



The HTML Canvas allows web application to draw graphics in real time, with functions to support drawing shapes, arcs, and text to a custom canvas element. Differences in font rendering, smoothing, anti-aliasing, as well as other device features cause devices to draw the image differently. This allows the resulting pixels to be used a part of a device fingerprint. The image on the left is a representative example of the types of canvas images used by fingerprinting scripts.

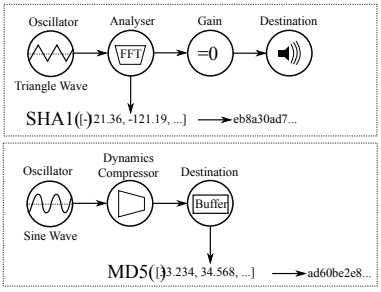
Findings

We found canvas fingerprinting on 14,371 sites, caused by scripts loaded from about 400 different domains. Comparing our results with those from our 2014 collaboration (https://securehomes.esat.kuleuven.be/%7Egacar/persistent/) with researchers at KU Leuven, we find three important trends. First, the most prominent trackers have by-and-large stopped using it, suggesting that the public backlash following that study was effective. Second, the overall number of domains employing it has increased considerably, indicating that knowledge of the technique has spread and that more obscure trackers are less concerned about public perception. Third, the use has shifted from behavioral tracking to fraud detection, in line with the ad industry's self-regulatory norm regarding acceptable uses of fingerprinting.

Full list of sites using Canvas Fingerprinting » (canvas_scripts.html)

Full script list (tsv) » (canvas_fingerprinting.tsv)

AudioContext Fingerprinting



Fingerprinting techniques typically aren't used in isolation but rather in conjunction with each other. By looking for unusual behavior in tracking scripts (e.g., use of new APIs) we found several fingerprinting scripts utilizing `AudioContext` and related interfaces. A manual analysis of these scripts suggest that trackers are attempting to utilize the Audio API to fingerprint users in multiple ways.

The figure on the right shows two different `AudioNode` configurations found during our study. In both configurations an audio signal is generated by an oscillator and the resulting signal is hashed after processing to create an identifier. This does not require access to the device's microphone, and instead relies on differences in the way the generated signal is processed. You can test your own device's Audio API fingerprint using our demonstration page here (https://webtap.princeton.edu/audio-fp).

Findings

In total, we found `AudioContext` fingerprinting of the type shown in the figure to the right in just 3 scripts present on 67 sites. Only two of these scripts appeared to be actively using the technique. Futher research is necessary to examine the stability and uniqueness of the fingerprint.

Full list of sites using AudioContext Fingerprinting » (audio_fp_scripts.html)

Full script list (tsv) » (audio_fingerprinting.tsv)

WebRTC Local IP Discovery

WebRTC is a framework for peer-to-peer Real Time Communication in the browser, and accessible via Javascript. To discover the best path between peers, each peer collects all available candidate addresses, including addresses from the local network interfaces (such as ethernet or WiFi) and addresses from the public side of the NAT and makes them available to the web application without explicit permission from the user. A fingerprinter can leverage these addresses to track users.

Findings

We found WebRTC being used to discover local IP addresses on 715 of the top 1 million sites. The vast majority of these instances were caused by third-party trackers.

Full list of Local IP Discovery scripts » (webrtc_scripts.html)

Full script list (tsv) » (webrtc_ip_retrieval.tsv)

Canvas-Font Fingerprinting

Javascript and Flash have both been used to enumerate fonts in the browser and use them to fingerprint users. The HTML Canvas API provides a third method to deduce the fonts installed on a particular browser. The canvas rendering interface exposes a `measureText` method, which provides the resulting width of text drawn to canvas. A script can attempt to draw text using a large number of fonts and then measure the resulting width. If the text's width is not equal to the width of the text using a default font (which would indicate that the browser does not have the tested font), then the script can conclude that the browser does have that font available.

Findings

In our measurement, we found canvas-based font fingerprinting on 3,250 first-party sites. A single third party (MediaMath) was responsible for the majority of font fingerprinting events, however a total of 5 other third parties were found to use the technique.

Full list of sites using Canvas-Font Fingerprinting » (font_scripts.html)

Full script list (tsv) » (font_fingerprinting.tsv)

Studies Using OpenWPM

For the list of studies that use OpenWPM please visit this page (https://webtap.princeton.edu/software/).

Data

The data is available as bzipped PostgreSQL dumps. The schema file used in all of the datasets is available here (data/schema.sql).

Dataset	Comme
1 Million Site Stateless	Parallel

(http://webtransparency.cs.princeton.edu/webcensus/data/census_2016_01_1m_stateless.sql.bz2)	Stateless Crawl
100k Site Stateful (http://webtransparency.cs.princeton.edu/webcensus/data/census_2016_01_100k_stateful.sql.bz2)	Parallel Stateful Crawl -- 10,000 si seed prof
10k Site ID Detection (1) (http://webtransparency.cs.princeton.edu/webcensus/data/census_2016_01_10k_id_detection_1.sql.bz2)	Sequentia Stateful Crawl -- Flash enabled - Synced w ID Detection (2)
10k Site ID Detection (2) (http://webtransparency.cs.princeton.edu/webcensus/data/census_2016_01_10k_id_detection_2.sql.bz2)	Sequentia Stateful Crawl -- Flash enabled - Synced w ID Detection (1)
55k Site Stateless with cookie blocking (http://webtransparency.cs.princeton.edu/webcensus/data/census_2016_01_blocking_block_cookies.sql.bz2)	Parallel Stateless Crawl -- Firefox se to block a third-par cookies
55k Site Stateless with Ghostery (http://webtransparency.cs.princeton.edu/webcensus/data/census_2016_01_blocking_ghostery.sql.bz2)	Parallel Stateless Crawl -- Ghostery extension installed and set to block all possible trackers
55k Site Stateless with HTTPS Everywhere (http://webtransparency.cs.princeton.edu/webcensus/data/census_2016_01_blocking_https_everywhere.sql.bz2)	Parallel Stateless Crawl -- HTTPS Everywhe installed

Code

The public repository for the OpenWPM crawling infrastructure is found on GitHub (https://github.com/citp/OpenWPM). The Princeton Web Census code is currently not public, but will be released in future iterations of the project.

Articles

Search Articles

CCPA

Consent Models

Consent Rates

Cookie Banners

Cookie Consent Features

Cookies 101

Are Cookies Used in Advertising?

Cookie Banner Guidelines for Each Global...

Cookie Law Definitions

How are Cookies Used?

What are Website Cookies?

What Cookies Does My Site Use?

What is a Tracking Pixel?

Last Updated: September 17, 2021

Tracking Pixel

A tracking pixel, also known as a *marketing pixel*, is a 1×1 pixel graphic used to track user behavior, site conversions, web traffic, and other metrics similar to a cookie. The tiny pixel-sided image is usually hidden and embedded in everything from banner ads to emails.

When implemented properly, these tiny bits of code can optimize your digital ads campaign and overall website. They will also increase your online conversion rate and help build an audience base.

The best example of a tracking pixel is one used by Google Analytics and similar services, which gathers data from websites. They can then tell those website owners the number of visitors and users that have seen their digital ads.

How Do Tracking Pixels Work?

If you have ever visited a website only to have ads from that business follow you to other sites and social media platforms, you are not alone. This common experience is possible through the use of tracking pixels.

You can add pixels by embedding them in your site's HTML code or email, which website, the and opens the

digital ad, or any e tracking pixel e data obtained d deliver

Types of Pixels

There are several different types of pixels in addition to a tracking pixel. Others include a conversion pixel or retargeting pixel. All help websites increase says by tracking marketing efforts. The information collected can help manage budgets and identify unnecessary costs in marketing campaigns.

Retargeting Pixel

Retargeting pixels are concerned mainly with the behavior of your website users. This type is of great use to digital marketers. Similar to our previous example, this is when a business's ad follows you to different sites and social media platforms.

Since these pixels depend on users that have already visited your site, they don't consistently produce high-volume campaigns. However, they do produce a better user experience by suggesting targeted and relevant content that can impact higher sales and encourage repeat customers.

Conversion Pixels

Conversion pixels are active once a purchase has actually been made. Specifically, they're responsible for tracking sales resulting from an ad campaign. When collecting data. Conversion pixels need to be placed within the code of the order confirmation page or email.

Most importantly, conversion pixels give digital marketers clarity into the source of their conversions and measure the success or failure of their marketing campaigns.

Facebook Pixel

A Facebook pixel is a tool for organizations using Facebook ads. The code is placed on your website and ensures that cookies track the users who interact with your Facebook ads and website.

This type of pixel is a unique code that collects data and allows websites to:

- Track Facebook ad conversions
- Build a target audience
- Optimize your ads
- Remarket to your website visitors

Learn how to integrate the Facebook pixel with a cookie banner script.

Differences Between a Pixel and Cookie

Tracking pixels and cookies are very similar and are often used simultaneously. They both serve similar marketing purposes by tracking user activity and behavior. However, the differences are in how the information is delivered and where it's kept.

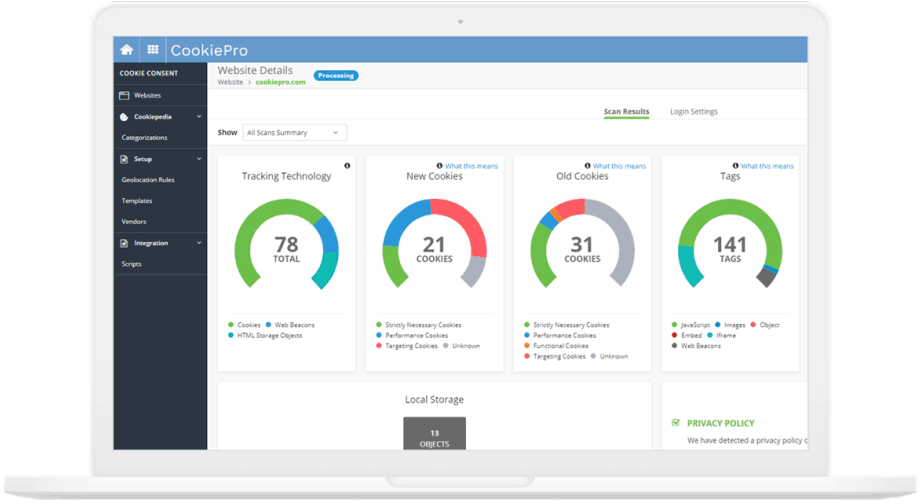
Cookies are dropped on a user's browser and can not follow them across devices. Additionally, users can block or clear cookies if they want. Most times, they're used to store user information for an easier login experience as well as adding multiple items to your cart for a single checkout experience.

Tracking pixels do not rely on the user's browser but will send information directly to servers. They can follow users across all of their devices which allow marketing efforts to be linked across website and mobile ads. A key difference is pixels cannot be disabled like cookies can.

Scan Your Website for FREE

CookiePro's FREE website scanner is the most powerful solution with flexible tracking and reporting capabilities. Start your scan today to uncover hidden cookies and tracking technologies such as pixels and web beacons on your website.

GET FREE SCAN



Get Started with CookiePro

evercookie

October 11, 2010: Reported on the front page of the New York Times

Find the latest details, code, and implementations on github @
https://github.com/samyk/evercookie

DESCRIPTION

evercookie is a javascript API available that produces
extremely persistent cookies in a browser. Its goal
is to identify a client even after they've removed standard
cookies, Flash cookies (Local Shared Objects or LSOs), and
others.

evercookie accomplishes this by storing the cookie data in
several types of storage mechanisms that are available on
the local browser. Additionally, if evercookie has found the
user has removed any of the types of cookies in question, it
recreates them using each mechanism available.

Specifically, when creating a new cookie, it uses the
following storage mechanisms when available:

- **Standard HTTP Cookies**
- **HTTP Strict Transport Security (HSTS) Pinning**
- **Local Shared Objects (Flash Cookies)**
- **Silverlight Isolated Storage**
- **Storing cookies in RGB values of auto-generated, force-cached PNGs using HTML5 Canvas tag to read pixels (cookies) back out**
- **Storing cookies in Web History**
- **Storing cookies in HTTP ETags**
- **Storing cookies in Web cache**
- **window.name caching**
- **Internet Explorer userData storage**
- **HTML5 Session Storage**
- **HTML5 Local Storage**
- **HTML5 Global Storage**
- **HTML5 Database Storage via SQLite**
- **HTML5 IndexedDB**
- **Java JNLP PersistenceService**
- **Java CVE-2013-0422 exploit (applet sandbox escaping)**

TODO: adding support for:

- Caching in HTTP Authentication
- Using Java to produce a unique key based off of NIC info
- Google Gears

Got a crazy idea to improve this? Email me!

EXAMPLE

Cookie found: *uid* = currently not set

Click to create an evercookie. Don't worry, the cookie is a random number between 1 and 1000, not enough for me to track you, just enough to test evercookies.

Click to create an evercookie

Now, try deleting this "uid" cookie anywhere possible, then

Click to rediscover cookies

or

Click to rediscover cookies WITHOUT reactivating deleted cookies

DOWNLOAD

evercookie is written in JavaScript and contains portions in Java, SWF/ActionScript (Flash) and C# (Silverlight). Some backend pieces in PHP, but also available in Node.js and Django.

Get the latest source from github: http://github.com/samyk/evercookie

FAQ

What is the point of evercookie?
Evercookie is designed to make persistent data just that, persistent. By storing the same data in several locations that a client can access, if any of the data is ever lost (for example, by clearing cookies), the data can be recovered and then reset and reused.

Simply think of it as cookies that just won't go away.

PRIVACY CONCERN! How do I stop websites from doing this?
Great question. So far, I've found that using Private Browsing in Safari will stop ALL evercookie methods after a browser restart.

What if the user deletes their cookies?
That's the great thing about evercookie. With all the methods available, currently thirteen, it only takes one cookie to remain for most, if not all, of them to be reset again.

For example, if the user deletes their standard HTTP cookies, LSO data, and all HTML5 storage, the PNG cookie and history cookies will still exist. Once either of those are discovered, all of the others will come back (assuming the browser supports them).

Why not use EFF's Panopticlick?
Panopticlick is an awesome idea, however the uniqueness really only helps in consumer machines and typically not systems running in a business or corporation. Typically those systems are virtually identical and provide no difference in information where a home user's laptop would. Evercookie is meant to be able to store the same unique data a normal cookie would.

Does this work cross-browser?
If a user gets cookied on one browser and switches to another browser, as long as they still have the Flash Local Shared Object cookie, the

Silverlight Isolated Storage, the Java JNLP PersistenceService or the Java CVE-2013-0422 exploit cookie, the cookie should reproduce in both browsers.

Does the client have to install anything?
No, the client simply uses the website without even knowing about the persistent data being set, just as they would use a website with standard HTTP cookies.

Does the server have to install anything?
The server must at least have access to the JavaScript evercookie file. Additionally, to use Local Shared Object (Flash Cookies) storage, the evercookie.swf file must be present, and to use the auto-generated PNG caching, standard caching and ETag storage mechanisms, PHP must be installed and evercookie_(png|etag|cache).php must be on the server.

All of these are available in the download.

Is evercookie open source?
Yes, evercookie is open source. The code is in readable format without any obfuscation. Additionally, the PHP files are open source as is the FLA (Flash) code used to generate the SWF Flash object. You can compile the Flash object yourself or use the pre-compiled version (evercookie.swf).

How does the PNG caching work?
When evercookie sets a cookie, it accesses evercookie_png.php with a special HTTP cookie, different than the one used for standard session data. This special cookie is read by the PHP file, and if found, generates a PNG file where all the RGB values are set to the equivalent of the session data to be stored. Additionally, the PNG is sent back to the client browser with the request to cache the file for 20 years.

When evercookie retrieves this data, it deletes the special HTTP cookie, then makes the same request to the same file without any user information. When the PHP script sees it has no information to generate a PNG with, it returns a forged HTTP response of "304 Not Modified" which forces the web browser to access its local cache. The browser then produces the cached image and then applies it to an HTML5 Canvas tag. Once applied, evercookie reads each pixel of the Canvas tag, extracting the RGB values, and thus producing the initial cookie data that was stored.

How does the Web History storage work
When evercookie sets a cookie, assuming the Web History caching is enabled, it Base64 encodes the data to be stored. Let's assume this data is "bcde" in Base64. Evercookie then accesses the following URLs in the background:

```
google.com/evercookie/cache/b
google.com/evercookie/cache/bc
google.com/evercookie/cache/bcd
google.com/evercookie/cache/bcde
google.com/evercookie/cache/bcde-
```

These URLs are now stored in history.

When checking for a cookie, evercookie loops through all the possible Base64 characters on google.com/evercookie/cache/, starting with "a" and moving up, but only for a single character. Once it sees a URL that was accessed, it attempts to brute force the next letter. This is actually extremely fast because **no requests** are made to theserver. The history lookups are simply locally in JavaScript using the CSS History Knocker. Evercookie knows it has reached the end of the string as soon as it finds a URL that ends in "-".

USAGE

```
<script type="text/javascript" src="evercookie.js"></script>

<script>
var ec = new evercookie();
```

```
// set a cookie "id" to "12345"
// usage: ec.set(key, value)
ec.set("id", "12345");

// retrieve a cookie called "id" (simply)
ec.get("id", function(value) { alert("Cookie value is " + value) });

// or use a more advanced callback function for getting our cookie
// the cookie value is the first param
// an object containing the different storage methods
// and returned cookie values is the second parameter
function getCookie(best_candidate, all_candidates)
{
    alert("The retrieved cookie is: " + best_candidate + "\n" +
            "You can see what each storage mechanism returned " +
            "by looping through the all_candidates object.");

    for (var item in all_candidates)
            document.write("Storage mechanism " + item +
                    " returned: " + all_candidates[item] + "<br>");
}
ec.get("id", getCookie);

// we look for "candidates" based off the number of "cookies" that
// come back matching since it's possible for mismatching cookies.
// the best candidate is most likely the correct one
</script>
```

SEE ALSO

csshack, best website ever

BUGS

See **CONTACT.**

CONTACT

Questions or comments, email me: code@samy.pl.

Visit http://samy.pl for more awesome stuff.

evercookie, by Samy Kamkar, 09/20/2010

Device Fingerprinting: Pros and Cons of the Controversial Method

Author: Nikolay Kryvtsun, Programmatic Specialist at Admixer

14 June 2021

Device fingerprinting, or you may have heard of it as web fingerprinting, mobile device fingerprinting, browser fingerprinting, or machine fingerprinting, has a concept similar to collecting human fingerprints by a detective, but not so obvious.

With 3rd-party cookies soon becoming obsolete, fingerprinting will be one of the **cookie alternatives** that will help marketers target users online. But will it last for long? Read on to find out.

Table of contents:

What is device fingerprinting?

Device fingerprinting is a technique, which analyses users' specific and unique configuration of software to determine their device or browser and, ultimately, track them online.

> Simply put, every time you connect to the web, you leave some hints behind about the device you're connected through. These hints are collected and used to create a profile about your device.

For fingerprinting the device, two things are needed: the operating system and the browser. Such information as your language preferences, time (down to milliseconds) your Internet Protocol (IP), screen resolution, and installed plug-ins, helps to analyze the profile of your device with machine fingerprinting.

How do marketers fingerprint devices?

Since fingerprinting does not require client-side storage of data, it is very difficult to notice and almost impossible to avoid. If 1st-party cookies are only valid within one domain, the unique characteristics remain unchanged when visiting different sites. This greatly simplifies tracking the user's movements on the Internet. Worse, unlike cookies, unique traits **cannot be disabled.** The user's efforts will lead to the maximum replacement of one set of features with another, even more recognizable.

For years, 3rd-party cookies were the main method for targeting people. HTTP cookies are small text files downloaded on your computer via browser during a stay on a website. Once you open the website in your browser, cookies are stored on your PC or Mac. The next time you visit it, you will get a more personalized experience. For instance, cookies provide information about your logins, items you put in a shopping cart, the size of your screen, and more.

3rd-party cookies will turn obsolete in 2022. Devices changed greatly, and third-party cookies are being replaced with methods, which provide more effective targeting. Moreover, cookies can be easily erased by a client and are recognized by ad blockers without difficulty. Marketers and advertisers can't continue using unreliable targeting services, so more of them choose device fingerprinting instead of cookies.

In contrast to web cookies that are located on the client-side in the user's browser, device fingerprinting is served on a server-side database.

Which data is collected with device fingerprinting

As we already stated, device fingerprinting is a method of gathering particular specifications of your device. It provides a wide range of data collection and sends it to a server after every request. Whatever you're doing on the web, device fingerprinting identifies the following:

- HTTP header

- Display resolution
- Browser type and version
- Device time
- Machine number
- Flash data
- Level of battery
- Mime types
- Operating system's information
- Variety of fonts
- Silverlight data
- IP address and much more information.

Pros: Device fingerprinting as an anti-fraud solution

The pros of device fingerprinting are that it helps to prevent online fraud. For instance, it can help to identify whether the Web banking session has been intercepted.

What is more, the technique allows to track fraudsters who steal login and payment card numbers. Proceeding transactions, they face the repeating process of employing trial and error methods. Fraudsters are just not able to change devices so often and can be tracked.

Cons: Data privacy concerns

Browser fingerprinting violates current privacy regulations as this way the users are not aware of the amount of their data transferred and who is getting hold of it. Users cannot simply clear their fingerprints like cookies, which poses concern even among the most privacy-conscious users.

Browsers are already working on fingerprinting restrictions and block some of the user specifications that are passed with the request. Besides, The World Wide Web Consortium (W3C) recently produced and published **notes and guidance for Web specification authors as to reduce the fingerprinting exposure**.

Thus, device fingerprinting is rather a temporary solution than a full-fledged cookie alternative and its capabilities will soon. It's clear that advertisers will soon be unable to use it to identify users.

Our advice for marketers is to focus on more privacy-forward identification solutions, including universal identifiers (like **Admixer ID**) and **user identity graphs**. They work with 1st-party user data obtained with proper consent and comply with effective privacy regulations.

Preventing device fingerprint: recommendations for users

If you are disturbed by the fact that your data is being collected and want to preclude its sharing through browser fingerprinting, you can stop it.

The only option to do it absolutely is to stop using the Internet at all, alas. We should admit that it's pretty much impossible to keep your browsers' data in disguise because HTTP headers are used in browsers to compile your device fingerprints.

But, it's not that bad.

Preventing device fingerprinting is not insurmountable. Besides, it is a very significant action to ensure safety while surfing the Internet. There are specific solutions, which make your personal web browsing information and your identity harder to trace.

1. Choosing the most popular browser

In this regard, you should forget about originality. Only by choosing the most popular browser, you can stay safe.

That is a great option to prevent device fingerprinting. You can't get easily targeted when you are among numerous browser users.

Moreover, not only should you use a popular browser, but also keep it updated. It's essential to download the latest version of your browser. Browsers are trying to keep their security solid and include improvements in each update, reducing the chances of becoming the target of device fingerprinting.

The latest version of the Mozilla Firefox browser claims to protect you against fingerprinting of devices. The browser blocks third-party requests to companies, which were caught red-handed in using device fingerprinting. Apple and Google also announced that they would restrict browser fingerprinting. Apple is going to hide the data, which can be fingerprinted, while Google made a draft of a "privacy budget" to limit the amount of data to fingerprint from the device by a certain company.

2. Private mode and incognito browsing

Another method to reduce device fingerprinting is to choose a private mode. This action doesn't stop the device fingerprinting completely, but it limits the general amount of data that is accessible to fingerprint.

In addition, we recommend the Tor browser for Incognito browsing, which is highly safe and confidential. Famous with dissidents in countries with censored Internet traffic. Tor browser has anti-fingerprinting features, it aims to make all users unidentified, reducing the possibility for you to be fingerprinted based on your browser and device information. The browser has the technology, which restricts the identification of your IP address. It hides your personal information, whether it's the language you prefer or the time zone you live in.

And we should make a small reminder that the most anonymous way to use the Internet browser is to exclude any plug-ins or extensions from the installed features. The one-of-a-kind fingerprint consists of all our extensions and plug-ins, not so many people choose the same ones.

Keeping your browser in the standard version is the best way to keep it untraceable.

3. Flash and Java-script disablement

Device fingerprinting applications mostly operate on Java-script or Flash. The great news is that browsers block it themselves, so in some way, it shouldn't bother you. You can check if your browser is still operating with any of them and disable or completely uninstall all of the features.

If not, your data will be tracked not so long because all of the major browsers have rejected the idea of supporting them.

4. VPN use

A virtual private network or shortened VPN increases your safety, privacy, and security. It routes your traffic by third-party servers masking your actual location. You seem like you're using the Internet in some other place, country, or even continent. VPN creates a virtual IP address and efficiently masks user's information. Therefore, a device fingerprint will include a fake IP address.

Similar to browsers, we recommend using a VPN service. The majority of people choose that. The more servers to connect with, the more difficult real information is to reach.

To wrap up

Device fingerprinting is affecting our online privacy significantly. The digital world is developing fast and causing challenges both for marketers and consumers. Because cookies are becoming obsolete and are losing their efficiency, device fingerprinting provides more powerful ways of targeting and collecting users' data. However, most device fingerprinting practices hardly comply with GDPR and other data protection regulations.

It is difficult to prevent the fingerprint of the device completely and almost impossible, but there are some ways to reduce the chances of a fingerprint. By the way, it's only up to a user whether to keep his data private and disable device fingerprinting or to allow it and stay available for a more personalized advertising experience.

Device fingerprinting is effective enough in stopping fraud and preventing suspicious payments. Also, the techniques of fingerprinting the device gained success in marketing and advertising.