**Technische Berichte in Digitaler Forensik**

**Herausgegeben vom Lehrstuhl für Informatik 1 der Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) in Kooperation mit dem Masterstudiengang Digitale Forensik(Hochschule Albstadt-Sigmaringen, FAU, Universität des Saarlandes)**

# Does the dark side still have (ever)cookies?

**Jonathan Schmidt**

06.03.2020

Technischer Bericht Nr. 18

## Zusammenfassung

Evercookie ist eine quelloffene JavaScript Bibliothek, die es ermöglichen soll, persistente Cookies in Browsern zu platzieren, indem verschiedene Speichermechanismen redundant benutzt werden. Laut Kommentaren auf GitHub können jedoch einige dieser Mechanismen nicht mehr verwendet werden. Dieser Bericht untersucht mit mehreren Browsern und in unterschiedlichen Szenarien, welche von Evercookie's Methoden noch funktionieren und welche nicht.

Entstanden im Rahmen des Konferenzseminars IT Sicherheit des Studiengangs Informatik im Wintersemester 2019/2020 unter der Anleitung von Gaston Pugliese.

**Hinweis:** Technische Berichte in Digitaler Forensik werden herausgegeben vom Lehrstuhl für Informatik 1 der Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) in Kooperation mit dem Masterstudiengang Digitale Forensik (Hochschule Albstadt-Sigmaringen, FAU, Universität des Saarlandes). Die Reihe bietet ein Forum für die schnelle Publikation von Forschungsergebnissen in Digitaler Forensik in deutscher Sprache. Die in den Dokumenten enthaltenen Erkenntnisse sind nach bestem Wissen entwickelt und dargestellt. Eine Haftung für die Korrektheit und Verwendbarkeit der Resultate kann jedoch weder von den Autoren noch von den Herausgebern übernommen werden. Alle Rechte verbleiben beim Autor. Einen Überblick über die bisher erschienen Berichte sowie Informationen zur Publikation neuer Berichte finden sich unter https://www1.cs.fau.de/df-whitepapers

# Does the dark side still have (ever)cookies?*

Jonathan Schmidt
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

## ABSTRACT

**Cookies were initially introduced to bring persistence to the stateless HTTP protocol. While first-party cookies can augment a website's functionality, the primary purpose of third-party cookies is tracking web users, mostly to provide tailored advertisement. In addition to HTTP cookies, there are several other storage mechanisms a website can use to place data persistently on client machines. To demonstrate how persistent, Samy Kamkar developed evercookie, a JavaScript library to set insistent cookies. It was released in 2010 and stores the data redundantly in as many places on the client's device as possible. Back then, it was hardly feasible for an inexperienced user to get rid of this cookie. However, since then, the web has evolved in many ways; browsers do provide more functionality to enhance privacy and issues on GitHub claim that evercookie's abilities are restricted. We present an introduction to the techniques used by evercookie to obtain this level of tenacity and test evercookie in multiple scenarios using different desktop and mobile browsers. Thereby, we provide an overview, which of evercookie's mechanisms still work under which circumstances and which not and why.**

## KEYWORDS

evercookie, cookies, web tracking, web privacy

## 1 INTRODUCTION

Cookies are required to enable essential features for modern web applications (e.g., Login sessions). However, they are largely employed to track users on the web [1, 6, 7]. Since we spend more and more time on the internet, our online activity uncovers a lot about our interests, our social life and our financial situation. This data is valuable for advertisers, insurance companies and banks [6]. To analyse users' online activity accurately, they need to be tracked over as many websites and as long as possible. Third-party cookies allow tracing users among multiple domains. For instance, studies by Englehardt and Narayanan discovered Google Analytics, the most prominent tracker, on almost 70% of the top 1 million web pages [7].

One option to address the persistence of cookies is evercookie, a JavaScript library released by Kamkar [24] in 2010. It utilises several storage mechanisms redundantly to preserve the cookie's information and resets it automatically whenever the user deletes some of them [19, 24]. Previous studies revealed that evercookie's methodology of using multiple storage mechanisms (e.g., HTTP cookies and Flash storage) redundantly is employed on 41 of the top 100 websites [1].

This paper is intended to provide an overview of how modern browsers deal with evercookies. Since it was presented back in 2010, the web has changed in many ways. The widespread HTML4 standard did not have any support for media or interactive content, which is why users had to install additional plugins like the Adobe Flash Player, Microsoft Silverlight or Java. With the introduction and adoption of HTML5 over the recent years and its native support for audio, video and interactive pages, those plugins became obsolete. No current web browser except the Internet Explorer supports Silverlight, and Adobe is going to stop supporting the Flash Player by the end of 2020 [28]. Furthermore, all common browsers discontinued the support for Java Applets between 2015 and 2017 [29, 34, 36, 37], and Oracle deprecates these applications since JDK 9 [35]. The exploits evercookie uses (Java CVE-2013-0422 [33] and CSS History Knocker [21]) have been known for almost a decade and should have been fixed for quite some time. Issues filed on GitHub [18] claim that evercookie would not be persistent anymore when cookies are removed [17] or when using the private browsing mode [11]. The cross-browser functionality is said to not work as well [15]. That leads to the questions of whether evercookie is still a useful tool to track users on the web and what can be done to stop it from working?

The paper is structured as follows: Section 2 explains the instrumented storage mechanisms and how evercookie utilises them. This includes Storage APIs, the browser's cache, HSTS Pinning and the CSS History Knocker. In Section 3 we describe, how our test setup works, which OS and browser combinations are instrumented and which scenarios are analysed. The results of the performed test are presented in Section 4. In Section 5 we discuss the test results and finally summarise the paper in Section 6.

The contributions of this paper are as follows:

(1) To the best of our knowledge, we performed the first extensive test of evercookie on both desktop and mobile browsers and demonstrated that evercookie is significantly limited in its abilities.
(2) We evaluated evercookie and present an analysis that shows why particular storage mechanisms are not accessible at all or are ineffective as persistent data storage, respectively.

## 2 BACKGROUND

In this section, we introduce the mechanisms applied by evercookie to set and maintain a tenacious cookie on client machines. Generally, evercookie uses all of the following techniques to store a single key-value pair redundantly [24]. Whenever the script detects that some data has been deleted, it tries to restore the value from another mechanism and reestablishes the removed data [19].

*This paper was written as part of the conference seminar "IT Security" which was organized by the chair of IT Security Infrastructures at the FAU during the winter term 2020. Special thanks to Gaston Pugliese for the provided support during the course of this paper.

## 2.1 HTTP cookies

The most trivial way for a website to store data on a client's device is an HTTP cookie [4]. An HTTP cookie is a key-value pair that can be set either by a Set-Cookie statement in the HTTP response header or by JavaScript [31]. Every time the client requests a website, the browser sends all cookies set by the requested domain in the HTTP header. Besides, cookies that are not marked as 'HTTP only' can also be accessed via JavaScript [4, 31].

## 2.2 Web storage APIs

The HTML5 standard defines four new client-side data stores. Local Storage, Session Storage, Web SQL and IndexedDB. Local Storage and Session Storage [23] are domain-specific databases that can store key-value pairs, which can be accessed using the corresponding JavaScript API. While the Session Storage gets deleted automatically when the browser window is closed, the Local Storage keeps information until it is removed by either the user or the website that set it.

Web SQL [22] and IndexedDB [2] are database interfaces, enabling websites to store more extensive amounts of data. While IndexedDB is available in all common browsers, Web SQL was deprecated by the World Wide Web Consortium (W3C) but is still available in Chromium-based browsers [22].

Evercookie would also support the HTML5 Global Storage, which no popular browser ever implemented, as it neglects the same-origin policy [6].

In Internet Explorer (IE), the userData Storage [27] is available. It was introduced with IE 5 and deprecated as of IE 7 [6], but still works under IE 11. It allows websites to store attributes within HTML elements persistently in the user agent.

If the plugins for Flash Player or Microsoft Silverlight are installed and enabled, a website is able to store data in the Local Shared Objects (LSO) Storage, also called Flash cookies, or the Silverlight Isolated Storage [26]. Both the LSO and the Isolated Storage are accessible for all browsers on a device. This empowers evercookie to set a cookie across different browsers by setting the remaining cookies based on the value from the LSO storage or the Isolated Storage, respectively [1, 6, 24].

Java Applets provided a similar interface: the Java Persistence Service. Since no customary browser supports Java Applets anymore [29, 34, 36, 37], evercookie can no longer use it.

Finally, evercookie renames the window.name property [8], which is a read and writable string for each tab that is resistant to page reloads. This enables cookie reviving if the website containing the evercookie is still open.

## 2.3 Cache

Browsers set up caches to accelerate page loads by avoiding redundant network traffic like static files. It is indiscernible for the website if a document was delivered by the requested server or the browser's cache. The HTTP protocol allows the client to indicate that there is no need to send a file if it has not been updated since a particular date (If-Modified-Since header). The server, on the other hand, can send a 304 response to the user agent, indicating that the requested document has not been changed since it was last delivered [9]. Evercookie uses the following techniques to place and retrieve data from the cache:

**Cached document:** First, evercookie sends a request with the cookie to be set to a PHP script that responds with a simple document that contains the data from the cookie and a Cache-Control header, telling the browser to cache the document. To retrieve the data, evercookie requests the same PHP script again, but without any request parameter. In this case, the server responds with 304 and thus, forces the browser to get the file from the cache. If it is present in the cache, the delivered document contains the original cookie value [6, 24].

**ETag:** Further, evercookie uses entity tags (ETags) [9] to store information in the cache. ETags were introduced as a cache-control technique, equipping a certain version of a resource with a unique identifier. When the browser wants to know, if a cached document is still up-to-date, it sends a conditional request, containing the ETag, to the server. The server can now determine if the cached resource is still the valid one and will only respond when the file has been updated. When evercookie sets a new cookie, it sends a request containing the data to a backend server which responds with a resource marked with the specified ETag. Later, the value can be recovered by demanding this file. The browser requests the document from the server and sends the ETag in the HTTP header. The server reads this ETag value and forwards it back to the client-side script. [6, 24]

**PNG image:** Finally, caching is used with pictures. Evercookie requests a PNG image with the cookie in the request arguments. The server-side script generates a PNG image where the characters of the cookie are written into the RGB values of the pixels. The generated image is sent back to the user agent where it is cached. To retrieve the image from the cache, evercookie requests the image source again without a cookie. The server responds with 304 to force the browser to load the image from its cache. Then, the HTML Canvas API is used to access the PNG file as an array of pixels, where evercookie can iterate over the pixels and thus, reconstructs the cookie value. [24]

## 2.4 HSTS Pinning

Cache mechanisms exist not only for website content but also for internal browser data. For instance, to remember which sites should be accessed encrypted using the HTTPS protocol, a server can send a Strict Transport Security header, forcing the browser to perform all future requests (till the expiration date) over HTTPS. This protocol is called HTTP Strict Transport Security (HSTS) [32]. Therefore, the browser stores the domains that sent an HSTS header in a previous request and should be accessed over HTTPS by default [32]. As a single domain can store only one bit of information, multiple domains are needed to enable tracking, as shown in Figure 1. When evercookie wants to store an n-bit integer, n domains are needed, all pointing to the same PHP file. The client-side script converts the integer into its bit representation and initiates requests to each domain, with the command to either set or delete this bit. When the script gets the set command, it responds with an HSTS header that has its expiration date far in the future. Otherwise, when it gets the delete command, it sends an HSTS header with the max-age attribute set to 0, which deletes an existing HSTS rule in the

browser. The integer can be reconstructed by requesting each domain. The server responds with the information if the request came over HTTP or HTTPS. Evercookie then combines the information from all domains and rebuilds the integer [3].

### 2.5 CSS History Knocker

The browsing history is not accessible for any website. However, Grossman [21] found a way to check if a particular site has been visited. First, the website places a link to the site to be checked. Then it reads the computed style of this link. Whenever the link is painted purple, instead of blue, the website is in the history [21]. When initialising, Evercookie accesses non-existent links in the pattern shown in Figure 2. When evercookie wants to read the cookie from the browser's history, it determines the cookie's value letter by letter. By iterating through all characters and digits, and checking if it matches an entry of the history, the first letter can be determined. This procedure is continued until the minus symbol is found, which indicates that the last letter was read [24].

### 2.6 Java Applet exploit

If Java Applets were still supported, evercookie could also take advantage of the Java CVE-2013-0422 exploit [33], which is a vulnerability in Java 7 Update 10 and earlier, allowing the Java application to abscond its sandbox and to access the clients file system [24].

## 3 METHODOLOGY

In this section, we explain the circumstances under which the tests were performed, define the browsers we instrumented and describe the scenarios we tested.

### 3.1 Test setup

To run the experiments, we use an NGINX server to host all the required documents, PHP scripts and to export our test results as shown in Figure 3. Further, we created a test page, containing the example of the index.html file from GitHub [19], which visualises the used storage mechanisms. Also, we added a text box to name the tested scenario and a button to export the result. When the button is clicked, a JavaScript function wraps the name together with the current values of each storage mechanism into a JSON string and sends it to the server. To efficiently evaluate our results, we implemented a Python script using the Flask framework, which processes the data from the client and writes it as a new line to a CSV file.

To enable HSTS Pinning, the server needs to support SSL. Therefore, we created an SSL certificate using OpenSSL and added it to the server. Besides, we added the corresponding root certificate to the macOS Keychain and the Windows Certificate Store. Thereby, all desktop browsers, except Firefox, trust the certificate and allow HTTPS communication without any warning. Firefox does not use the OS, but its own database to validate certificates, but allows us to add exceptions for our local domains manually [30].

### 3.2 Instrumented browsers

Our survey on evercookie should be as comprehensive as possible. That is why we chose to test the browsers with the highest market share according to StatCounter [38, 39], each of them in their latest published version. This results in Google Chrome Version 78, Mozilla Firefox Version 71, Apple Safari Version 13, Microsoft Edge Version 44 and Microsoft Internet Explorer 11 on the desktop side. Chrome, Safari and Firefox are running on macOS 10.15, Edge and Internet Explorer are running on Microsoft Windows 10 Version 1909. On the mobile side, Chrome for Android Version 78 and Safari for iOS are employed. Chrome for Android is operating on a Samsung Smartphone running Android 9, Safari for iOS is running on an Apple iPhone with iOS 13.3.

### 3.3 Scenarios & storage mechanisms

As mentioned in Section 2, evercookie cannot use particular storage mechanisms as the browsers removed the support. More specifically, HTML Global Storage, Java Persistence Service and the Java Applet exploit [33] are not supported by any of the reviewed browsers, and we see no need to test them.

Apart from that, all supported storage mechanisms are used. More precisely: HTTP cookies, Local and Session Storage, Web SQL, IndexedDB, Internet Explorer userData, the window.name property, Local Shared Objects, Silverlight Isolated Storage, cache data, ETag, PNG image, History and HSTS Pinning.

To test HSTS Pinning, we modify the hosts file of the OS so that it resolves the domains {a-j}.ec to the local IP address where our server with the corresponding PHP script is running. Since the hosts file is only accessible on desktop operating systems, at least one public domain, including a trusted SSL certificate, would be necessary to test HSTS Pinning on mobile devices. Since we do not have this infrastructure available, we leave the testing of HSTS Pinning on mobile devices to future work.

Issues filed on GitHub [18] claim that evercookie does not have the persistence anymore, that one would expect from its name. The most relevant and repetitive issues relate to the persistence when clearing the history [13, 17], private mode [11], the cross-browser functionality [15], Flash Storage (LSO) [14] and the CSS History Knocker [16]. From this, we derive the following test cases:

**Default settings:** Here, we use the browsers freshly installed and with no settings changed. After we created an evercookie, we monitor the storage mechanisms it uses. Afterwards, we delete all cookies with the standard tool the browsers offer and examine if the evercookie can be restored.

**Private mode:** In this scenario, we set an evercookie within the private mode (Incognito mode in Chrome). Unlike in the default mode, the expectation is, that all cookies are gone once the window is closed. Manual removal of any browsing data should not be necessary.

**Flash Player + Silverlight enabled:** This test is executed only on the desktop side, as mobile browsers support none of the plugins. The Flash Player is available in all browsers, whereas Silverlight is only available in Internet Explorer. In this scenario, we use the default browsing mode again and do not have installed any other plugins or extensions. After we created a new evercookie, we observe, which storage mechanisms are in use and delete the cookies using the standard tools again.

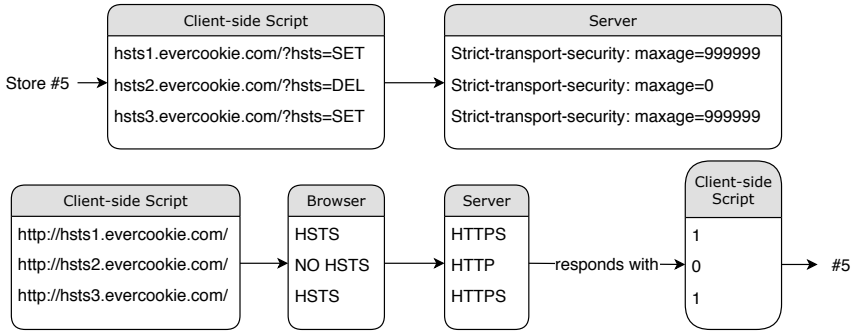


**Figure 1: Storing an identifier (here: an integer with the value of 5) by using HSTS across multiple domains [3]**

```
google.com/evercookie/cache/v
google.com/evercookie/cache/va
google.com/evercookie/cache/val
google.com/evercookie/cache/valu
google.com/evercookie/cache/value
google.com/evercookie/cache/value-
```

**Figure 2: Pattern of accessing websites in the background to efficiently use the History Knocker to read a cookie from the browsing history [24]**

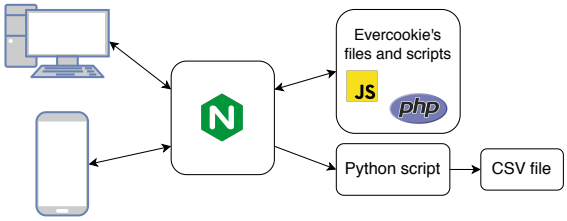


**Figure 3: Our test setup: An NGINX server for evercookie's backend and a Python script to export the results**

## 4 RESULTS

In this section, we present the outcomes of our tests. The results of our evaluation are described below and depicted in Table 1.

### 4.1 Default settings

**Desktop:** When we create an evercookie, it can use the following storage mechanisms in all browsers. HTTP cookies, Local- and Session Storage, IndexedDB, the window.name property, the cached document and HSTS Pinning. Chrome, Firefox and Safari support all three cache mechanisms. In contrast, Internet Explorer does only support the cached document and PNG image. Edge only maintains the ETag beside the cached document. Chrome additionally supports the Web SQL interface and allows evercookie to use it as well. Although Internet Explorer has support for it, the userData storage is not in use.

Next, it is about deleting the website data with the tools the browsers offer. Clearing browsing data while the tab with the evercookie is still open, removes the information from all mechanisms, except the window.name attribute. This allows evercookie to revive the other cookies in all browsers. However, when the tab has been closed before clearing all website data, evercookie gets deleted reliably, except in Firefox, where offline website data (IndexedDB) remains by default when the history is cleared.

**Mobile:** On our mobile devices, we can see quite similar results as on the desktop side. In both Safari for iOS and Chrome for Android, evercookie can use conventional HTTP cookies, Local- and Session Storage, all cache mechanisms, HSTS Pinning and the window.name attribute. In Chrome for Android, we have support for Web SQL additionally. When we clear the browsing data, after we close the tab, the evercookie gets annihilated on both iOS and Android. However, if we do not, it can restore the value through the window.name attribute, again.

### 4.2 Private mode

**Desktop:** When using the private browsing mode, we can see only minor differences compared to the default mode. Firefox, Edge and Internet Explorer disable the support for the IndexedDB, and the cached PNG image is not accessible in the Internet Explorer anymore. Besides, evercookie is still able to instrument the same storage mechanisms as when using a conventional window. Nevertheless, after the private window has been closed, evercookie cannot restore its value in any browser in neither the default mode nor a new private window.

**Mobile:** The same applies to our mobile browsers. Evercookie can make use of the same storage mechanisms as in the default mode, but cannot restore the cookie value from any storage after we closed the private window.

### 4.3 Flash and Silverlight enabled

Although all reviewed desktop browsers do support Flash, evercookie only manages to embed data in the Local Shared Objects in Firefox and Internet Explorer. Details, why evercookie could not use the LSO in Chrome, Safari and Edge, are given in Section 5.3.

Next, we delete the browsing data again and observe if Flash cookies are getting removed as well. The Internet Explorer clears the LSO Storage, and the evercookie gets deleted. Firefox, however, does not and thus, enables evercookie to restore the cookie from the Flash storage.

### 4.4 Cross-browser functionality

First, the cross-browser synchronisation is technically only possible among desktop browsers, as it requires at least one of the plugins for Flash, Silverlight or Java. Neither Android nor iOS has support for them. Since we did not test Java Applets and evercookie could not enter the Silverlight Isolated Storage, there is only the Flash plugin left to enable a cross-browser cookie. Evercookie manages to enter the LSO in Firefox and Internet Explorer. Thus, we run the test on Windows using Firefox and Internet Explorer. We reset both browsers, set a new evercookie in Firefox, click on restore in the Internet Explorer and observe that evercookie can read the value from the Local Shared Objects and sets the remaining mechanisms appropriately.

## 5 DISCUSSION

In this section, we discuss the results of our tests that we presented in Section 4. We evaluate the results separately for each tested scenario.

### 5.1 Default settings

When we analyse our results, the essential observation we make is that except for Firefox, no browser is vulnerable to evercookie. Firefox does not delete the IndexedDB by default, but this can be changed via an option in the clear history dialogue. All browsers deleted HTTP cookies, the history, all data stored through Web Storage APIs, the cache and the HSTS rules.

A surprising fact is that HSTS Pinning works well in Safari, although WebKit, the engine behind Safari, implements a feature to prevent this. By accepting HSTS headers only for the current sub-domain and for the current second-level domain, there is a maximum of two HSTS rules that a single site should be able to set [10].

Although Internet Explorer has support for the userData Storage, evercookie cannot utilise it. We examined the corresponding code with the debugger and observed that the `addBehavior` function, which is necessary to store information, is unavailable.

The CSS History Knocker does not work in any browser. The `getComputedStyle` function now always returns the default colour, no matter if the link was visited or not, making it impossible for evercookie to determine the previously visited pages.

Finally, we want to examine, why the PNG image mechanism is not accessible in Edge, and why the ETag mechanism is not usable in Internet Explorer. For some reason, the Edge browser does not request the PNG image and thus, the `onload` function, in which the cookie value gets extracted from the image, is never called. The ETag mechanism is unavailable in Internet Explorer, as the cache of the Internet Explorer does not validate the topicality of a resource within a browsing session [25]. However, this is required, as only the server can read the ETag.

Overall, evercookie is not a significant threat when the browsers are used with default settings, and the user knows how to delete browsing data.

### 5.2 Impact of private mode

Next, we want to examine if the private mode impacts the effectiveness of evercookie. In short: The private mode does, what it is supposed to do: Clearing all data of a browsing session once the window is closed. None of the browsers allowed evercookie to recover any data in the default mode or a new private window. One interesting point is, that Firefox, Edge and Internet Explorer disable the support for the IndexedDB interface when a website is visited in private mode, as this API is meant as long term storage for offline data.

According to evercookie's GitHub page [19], the HSTS Pinning should have worked in the Incognito mode of Chrome. That is because previous versions of Chrome had only a single file to store all HSTS rules, which were adopted when switching to the private mode. Though, this has been changed so that Chrome creates an isolated HSTS rule set for the private browsing session [5].

The other difference is that evercookie cannot set data through the cached PNG image in the Internet Explorer for the same reason as it is in Edge in default mode. The PNG image is never requested although its source is specified.

### 5.3 Plugins enabled

First, we want to discuss why evercookie could not manage to set a cookie in the LSO in Chrome, Safari and Edge. Evercookie uses SWFObject [20] to embed Flash content into a web page. The developers of this open-source library stopped maintaining it six years ago, and its JavaScript code is in an obfuscated, unreadable format, making investigations complicated. The browsers are set to ask before Flash content is launched, but none of them displayed any message asking if Flash content should be executed, which indicates that SWFObject could not manage to start the Flash application. Since Adobe is going to drop the support for Flash by the end of 2020, we do not see any reason for further research here.

Next, we analyse how the two browsers Firefox and Internet Explorer, where evercookie could embed data in the Local Shared Objects, deal with it when we clear the cookies. We already observed that Internet Explorer removes Flash cookies when we delete the history. However, Firefox does not, and thus, enables cookie reviving not only through the IndexedDB but also through the LSO storage. Again, Firefox is able to delete Flash cookies and the IndexedDB, but only after manual configuration.

Finally, we examine why evercookie cannot use the Silverlight Isolated storage in the Internet Explorer. Therefore, we use the debugger and execute the function, where Silverlight is initialised and embedded, line by line. This results in an exception when the Silverlight HTML object gets appended to the body of the page.

| Scenario | Browser | OS | HTTP cookie | Local Storage | Session Storage | IndexedDB | Web SQL | userData | window.name | Local Shared Objects | Isolated Storage | Cache | ETag | PNG image | History | HSTS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Default | Chrome 78 | macOS 10.15 | ○ | ○ | ○ | ○ | ○ | – | ◑ | – | – | ○ | ○ | ○ | – | ○ |
| | Firefox 71 | macOS 10.15 | ○ | ○ | ○ | ● | – | – | ◑ | – | – | ○ | ○ | ○ | – | ○ |
| | Safari 13 | macOS 10.15 | ○ | ○ | ○ | ○ | – | – | ◑ | – | – | ○ | ○ | ○ | – | ○ |
| | Edge 44 | Windows 10 | ○ | ○ | ○ | ○ | – | – | ◑ | – | – | ○ | ○ | – | – | ○ |
| | Internet Explorer 11 | Windows 10 | ○ | ○ | ○ | ○ | – | – | ◑ | ○ | – | ○ | – | ○ | – | ○ |
| | Safari 13 | iOS 13 | ○ | ○ | ○ | ○ | – | – | ◑ | – | – | ○ | ○ | ○ | – | ✖ |
| | Chrome 78 | Android 9 | ○ | ○ | ○ | ○ | ○ | – | ◑ | – | – | ○ | ○ | ○ | – | ✖ |
| Private mode | Chrome 78 | macOS 10.15 | ○ | ○ | ○ | ○ | ○ | – | ○ | – | – | ○ | ○ | ○ | – | ○ |
| | Firefox 71 | macOS 10.15 | ○ | ○ | ○ | – | – | – | ○ | – | – | ○ | ○ | ○ | – | ○ |
| | Safari 13 | macOS 10.15 | ○ | ○ | ○ | ○ | – | – | ○ | – | – | ○ | ○ | ○ | – | ○ |
| | Edge 44 | Windows 10 | ○ | ○ | ○ | – | – | – | ○ | – | – | ○ | ○ | – | – | ○ |
| | Internet Explorer 11 | Windows 10 | ○ | ○ | ○ | – | – | – | ○ | ○ | – | ○ | – | – | – | ○ |
| | Safari 13 | iOS 13 | ○ | ○ | ○ | ○ | – | – | ○ | – | – | ○ | ○ | ○ | – | ✖ |
| | Chrome 78 | Android 9 | ○ | ○ | ○ | ○ | ○ | – | ○ | – | – | ○ | ○ | ○ | – | ✖ |
| Flash + Silverlight enabled | Chrome 78 | macOS 10.15 | ○ | ○ | ○ | ○ | ○ | – | ◑ | – | – | ○ | ○ | ○ | – | ○ |
| | Firefox 71 | macOS 10.15 | ○ | ○ | ○ | ● | – | – | ◑ | ● | – | ○ | ○ | ○ | – | ○ |
| | Safari 13 | macOS 10.15 | ○ | ○ | ○ | ○ | – | – | ◑ | – | – | ○ | ○ | ○ | – | ○ |
| | Edge 44 | Windows 10 | ○ | ○ | ○ | ○ | – | – | ◑ | – | – | ○ | ○ | – | – | ○ |
| | Internet Explorer 11 | Windows 10 | ○ | ○ | ○ | ○ | – | – | ◑ | ○ | – | ○ | – | ○ | – | ○ |

**Table 1: Comprehensive overview of the tested scenarios, browsers and which storage mechanisms are used**
**–: evercookie cannot use the storage mechanism, see Section 5 for details**
**○: evercookie uses the storage, but it gets cleared on cookie deletion (In private mode: when the window is closed)**
**◑: storage remains as long as the page containing the evercookie is still open**
**●: evercookie recovers itself from the storage, although cookies were removed (In private mode: after the window was closed)**
**✖: not tested**

This bug has already been reported on GitHub but does not seem to be fixed so far [12].

### 5.4 Cross-browser capability

The cross-browser functionality is only possible when Flash is enabled and working. If evercookie had managed to use the Flash storage in the other browsers as well, the cross-browser synchronisation would have worked among more browsers. By the end of 2020, when Flash reaches its end of support and all browsers will remove the plugin, the cross-browser functionality will stop working entirely.

## 6 CONCLUSION

We tested evercookie with modern desktop and mobile browsers and found out that evercookie is still capable of using nearly all of its original storage mechanisms. Despite, it still does not work reliably in any of the analysed browsers and is not persistent in private browsing mode. There are two essential reasons for this: First, browsers do not only delete HTTP cookies and the browsing history but also clear the cache, HSTS rules and all data stored through web storage APIs. Second, the plugins for Flash, Silverlight and Java are not necessary anymore, since the HTML5 standard provides all of their functionality. By the end of 2021, Silverlight will be the last of these plugins to reach its end of support and thus, will limit evercookie's abilities even more. This development, combined with minor changes in Firefox's default settings, would cause evercookie to be entirely ineffective within the reviewed browsers.

## REFERENCES


[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 674–689.

[2] Ali Alabbas and Joshua Bell. 2018. Indexed Database API 2.0. https://www.w3.org/TR/IndexedDB-2/ visited on December 30, 2019.

[3] Zbigniew Banach. 2019. Why Websites Need HTTP Strict Transport Security (HSTS) | Netsparker. https://www.netsparker.com/blog/web-security/http-strict-transport-security-hsts/ visited on December 2, 2019.

[4] A. Barth. 2011. RFC 6265 - HTTP State Management Mechanism. https://tools.ietf.org/html/rfc6265 visited on December 30, 2019.

[5] Chromium bugs. 2017. 774643 - Clearing non-incognito data results in retainining history in incognito's TransportSecurityPersister. https://bugs.chromium.org/p/chromium/issues/detail?id=774643 visited on January 4, 2020.

[6] Tomasz Bujlow, Valentín Carela-Español, Josep Sole-Pareta, and Pere Barlet-Ros. 2017. A survey on web tracking: Mechanisms, implications, and defenses. *Proc. IEEE* 105, 8 (2017), 1476–1510.

[7] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 1388–1401.

[8] Steve Faulkner, Arron Eichholz, Travis Leithead, Alex Danilo, and Sangwhan Moon. 2017. HTML 5.2: 6. Loading Web Pages. https://www.w3.org/TR/html52/browsers.html#dom-window-name visited on December 30, 2019.

[9] R. Fielding and J. Reschke. 2014. RFC 7232 - Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. https://tools.ietf.org/html/rfc7232 visited on December 30, 2019.

[10] Brent Fulgham. 2018. Protecting Against HSTS Abuse | WebKit. https://webkit.org/blog/8146/protecting-against-hsts-abuse/ visited on January 4, 2020.

[11] GitHub. 2012. Evercookie doesn't work with Private Browsing mode - Issue #19 - samyk/evercookie. https://github.com/samyk/evercookie/issues/19 visited on November 29, 2019.

[12] GitHub. 2013-2015. document.body.appendChild throws error - Issue #45 - samyk/evercookie. https://github.com/samyk/evercookie/issues/45 visited on January 4, 2020.

[13] GitHub. 2014 - 2015. if i clean cookies in chrome evercookie forget me - Issue #81 - samyk/evercookie. https://github.com/samyk/evercookie/issues/81 visited on November 29, 2019.

[14] GitHub. 2016. Clearing History/Browsing Data - No More Persistence - Issue #118 - samyk/evercookie. https://github.com/samyk/evercookie/issues/118 visited on November 29, 2019.

[15] GitHub. 2016. cross browser not working - Issue #119 - samyk/evercookie. https://github.com/samyk/evercookie/issues/119 visited on November 29, 2019.

[16] GitHub. 2016. CSS history knocking / leak fixed? - Issue #115 - samyk/evercookie. https://github.com/samyk/evercookie/issues/115 visited on November 29, 2019.

[17] GitHub. 2017 - 2018. Evercookie is not persistent in Chrome and Safari anymore - Issue #125 - samyk/evercookie. https://github.com/samyk/evercookie/issues/125 visited on November 29, 2019.

[18] GitHub. 2019. Issues - samyk/evercookie. https://github.com/samyk/evercookie/issues visited on November 29, 2019.

[19] GitHub. 2019. samyk/evercookie. https://github.com/samyk/evercookie visited on November 29, 2019.

[20] GitHub. 2019. swfobject/swfobject. https://github.com/swfobject/swfobject visited on November 29, 2019.

[21] Jeremiah Grossman. 2006. Jeremiah Grossman: I know where you've been. https://blog.jeremiahgrossman.com/2006/08/i-know-where-youve-been.html visited on December 4, 2019.

[22] Ian Hickson. 2010. Web SQL Database. https://www.w3.org/TR/webdatabase/ visited on December 30, 2019.

[23] Ian Hickson. 2016. Web Storage (Second Edition). https://www.w3.org/TR/webstorage/ visited on December 30, 2019.

[24] Samy Kamkar. 2010. Samy Kamkar - Evercookie. https://samy.pl/evercookie/ visited on November 29, 2019.

[25] Eric Lawrence. 2010. Caching improvements in Internet Explorer 9 | Microsoft Docs. https://docs.microsoft.com/en-gb/archive/blogs/ie/caching-improvements-in-internet-explorer-9 visited on January 4, 2020.

[26] Microsoft. 2011. Isolated Storage | Microsoft Docs. https://docs.microsoft.com/en-us/previous-versions/windows/silverlight/dotnet-windows-silverlight/bdts8hk0(v=vs.95) visited on December 22, 2019.

[27] Microsoft. 2013. userData Behavior | Microsoft Docs. https://docs.microsoft.com/en-us/previous-versions/ms531424(v%3Dvs.85) visited on January 4, 2020.

[28] Microsoft. 2019. Adobe Flash end of support. https://support.microsoft.com/de-de/help/4520411/adobe-flash-end-of-support visited on December 10, 2019.

[29] Microsoft. 2019. Microsoft Edge - Frequently Asked Questions. https://docs.microsoft.com/en-us/microsoft-edge/deploy/microsoft-edge-faq visited on December 20, 2019.

[30] Mozilla. 1998 - 2020. Mozilla CA Certificate Store - Mozilla. https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/ visited on January 28, 2020.

[31] Mozilla and individual contributors. 2005 - 2019. Document.cookie - Web APIs | MDN. https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie visited on December 2, 2019.

[32] Mozilla and individual contributors. 2005-2019. Strict-Transport-Security - HTTP | MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security visited on December 2, 2019.

[33] NIST. 2013. NVD - CVE-2013-0422. https://nvd.nist.gov/vuln/detail/CVE-2013-0422 visited on December 4, 2019.

[34] Oracle. [n. d.]. Java and Apple Safari Browser. https://java.com/en/download/faq/safari.xml visited on December 20, 2019.

[35] Oracle. 2017. java.applet (Java SE 9 & JDK 9). https://docs.oracle.com/javase/9/docs/api/java/applet/package-summary.html visited on December 4, 2019.

[36] Justin Schuh. 2014. Chromium Blog: The Final Countdown for NPAPI. https://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html visited on December 20, 2019.

[37] Benjamin Smedberg. 2015. NPAPI Plugins in Firefox - Future Releases. https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/ visited on December 20, 2019.

[38] StatCounter. 2019. Desktop Browser Market Share Worldwide | StatCounter Global Stats. https://gs.statcounter.com/browser-market-share/desktop/worldwide visited on November 29, 2019.

[39] StatCounter. 2019. Mobile Browser Market Share Worldwide | StatCounter Global Stats. https://gs.statcounter.com/browser-market-share/mobile/worldwide visited on November 29, 2019.