Independent Study - Survey of Web Tracking and Defence

Name: NG, Yiu Wai
Date: May 2019

Table of Contents

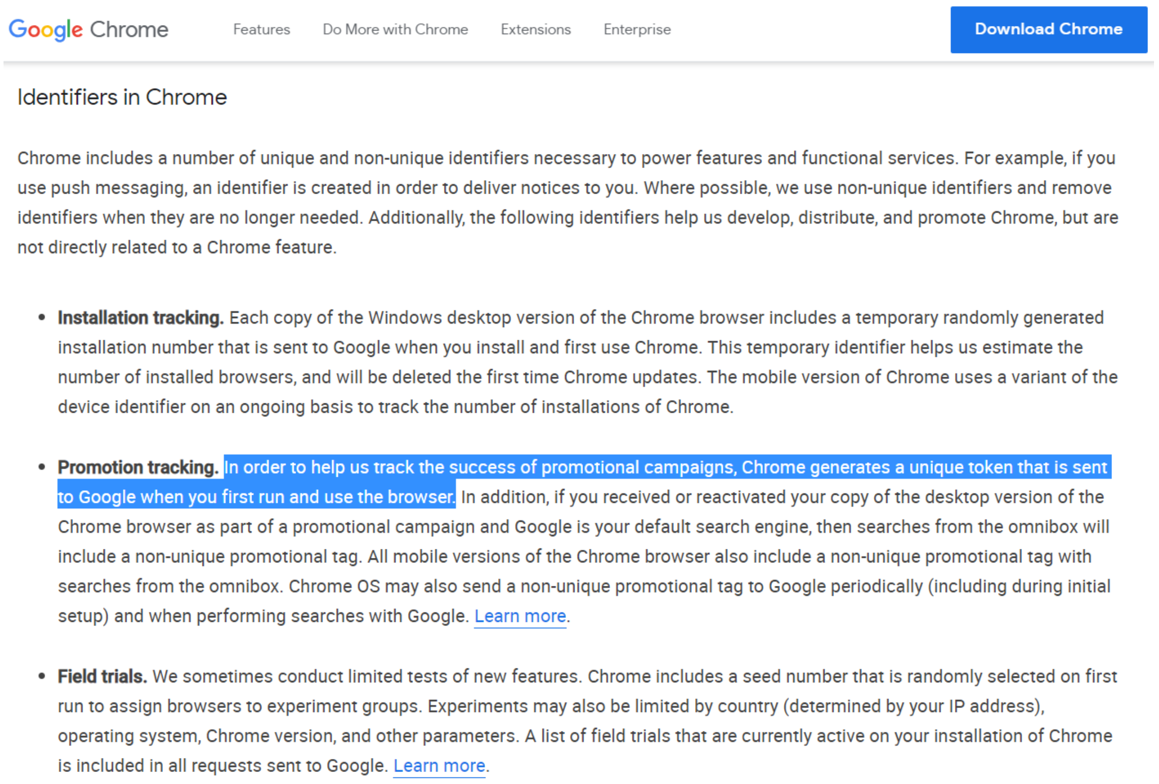
■ Introduction: Web Tracking: Why, Who and How?

Third-party web tracking is the practice by which entities ("trackers") embedded in webpages re-identify users as they browse the web, collecting information about users' activities on websites[1]. Web tracking has its pros and cons. It is arguable that should web tracking be prohibited.

Advertising companies may collect web users' browsing records to suggest products which users needed. Host of websites may collect visitor's browsing activities to improve the webpages design. On the other side, users are afraid that their personal data are collected and leaked to dishonest parties.

Different parties have different viewpoints on web tracking. However, everyone agree that users should have the right to choose. Users should have the right to choose what kinds of information are provided to which parties.

Researchers and regulators have done a number of works to understand web tracking. Policies are imposed to allow users to choose. In a research in year 2016[2], it shows that *Google* is the top third parties tracker in the internet. The tracker's domains, *google-analytics.com*, *gstatic.com*, and *doubleclick.net*, appear in most websites. The social media websites, *Facebook* and *Twitter,* are also trackers commonly found in websites.



Identifiers in Chrome

Chrome includes a number of unique and non-unique identifiers necessary to power features and functional services. For example, if you use push messaging, an identifier is created in order to deliver notices to you. Where possible, we use non-unique identifiers and remove identifiers when they are no longer needed. Additionally, the following identifiers help us develop, distribute, and promote Chrome, but are not directly related to a Chrome feature.

- **Installation tracking.** Each copy of the Windows desktop version of the Chrome browser includes a temporary randomly generated installation number that is sent to Google when you install and first use Chrome. This temporary identifier helps us estimate the number of installed browsers, and will be deleted the first time Chrome updates. The mobile version of Chrome uses a variant of the device identifier on an ongoing basis to track the number of installations of Chrome.
- **Promotion tracking.** In order to help us track the success of promotional campaigns, Chrome generates a unique token that is sent to Google when you first run and use the browser. In addition, if you received or reactivated your copy of the desktop version of the Chrome browser as part of a promotional campaign and Google is your default search engine, then searches from the omnibox will include a non-unique promotional tag. All mobile versions of the Chrome browser also include a non-unique promotional tag with searches from the omnibox. Chrome OS may also send a non-unique promotional tag to Google periodically (including during initial setup) and when performing searches with Google. Learn more.
- **Field trials.** We sometimes conduct limited tests of new features. Chrome includes a seed number that is randomly selected on first run to assign browsers to experiment groups. Experiments may also be limited by country (determined by your IP address), operating system, Chrome version, and other parameters. A list of field trials that are currently active on your installation of Chrome is included in all requests sent to Google. Learn more.

Figure 1: Default user identifying policy in Google Chrome Browser[3]. Google use insert unique and non-unique identifiers when Chrome Browser sends requests. This can be used for tracking user's activities.

Researcher summarized common scenarios of how third party obtain user's identity[4]:

(1) The third party is also a first party

Third party is a first party in another context, where the users provides their identity voluntarily. For example, the Facebook "Like" button embedded in webpages.

(2) A first party sells the user's identity to third party

Third party may purchase user's identity from first party. Some advertising data providers buy identifying information from first party, and use it to target advertising.

(3) A first party unintentionally provides identity to third party

First party may unintentionally leak user identity to third party because the website uses unsecured data transmission method. For example, the website puts identifying information in a URL or page title, which allows third party to access.

(4) The third party uses a security exploit

Third party may exploit a cross-site vulnerability on a first party website to steal user's identity. Users unintentionally grant authorization to a third party to access their browsing histories.

(5) Re-identification

Third party may match browsing histories against identified datasets to re-identify users.

For (1) and (2), users are tracked because a trusted first party gives users identifying information to third parties. It is a problem on privacy protection policy, and some jurisdictions have imposed laws to regulate such activities.

For (3), users are tracked because of the poor design of webpages. Data are transmitted in unsecured channels. Host of websites should regularly review their website and ensure only authorized parties can obtain users' identifying information.

For (4), users are tracked because they unintentionally authorized trackers to obtain their identifying information. Trackers launch web attacks to steal identifying information.

For (5), users are tracked because trackers re-identify users by matching publicly available information (e.g. IP, browser type, browsing time, …) in the Internet.

In this survey, researches on stateful web tracking are studied and summarized. Issues related to scenarios (1) to (4) are discussed, including

- How do first party track users: identifying cookies
- How do third party track users: get cookies using XSS or CSRF
- Current approaches for defensing stateful web tracking

■ Problems of Stateful Tracking: Identifying Cookies

HTTP Cookie: What is it?

The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypermedia information systems[5]. Each time a user clicks on a hypertext link in a web browser, the browser typically connects to the web server and sends it a request message using HTTP. After the web browser received a response, it disconnects from the web server. Each request from user creates a new connection between web browser and web server. Therefore, HTTP request is considered as “stateless”: each request is treated completely independent from previous one.[6]

The “stateless” characteristic is not desirable for some web applications. For example, a website may allow users to login and access restricted information. Under this stateless connection, browser needs to include the full identifying information (e.g. User ID plus password) each time before sending HTTP request to web server. Web server needs to verify the identifying information from user each time before sending a respond. It is inconvenient for everyone. Hence, HTTP cookie is introduced to make HTTP “stateful”.

HTTP Cookie, or *Set-Cookie* header fields, is a header fields used by HTTP servers to store state (called cookie) at HTTP user agents, letting the servers maintain a stateful session over the mostly stateless HTTP protocol.[7] It is a small piece of information that the web server and web browser pass back and forth for identifying a user in each stateless connection.

HTTP Cookie: How trackers use it?

HTTP Cookie is a piece of information locally stored at user’s browser. It is designed to solve the inconvenience of stateless connection. Users can be identified in each session of browsing. However, web server may enhance the cookies to identify users in multiple sessions for the purpose of tracking users’ browsing activities. Researcher studied this “identifying cookie” by analysing cookies with below characteristics[8]:

(1) The cookie is *long-lived*, with long expiry period.

(2) The cookie’s value is *stable*, and remains constant through all page visits.

(3) The cookie has a *constant-length* across all datasets.

(4) The cookie is *user-specific*, so the value is unique across different browser instances.

(5) The cookie has *high-entropy* value string, with values sufficiently difference between machines to enable unique identification.

The cookies are unique and remain existed after the browsing sessions ended. Tracker can re-identify a user when the user’s browser connected to the web server again.

Third Party Tracking: How Third-Party Get Your HTTP Cookies?

First party can track a user by tailor-made cookies. If users allow a website to create cookies in web browser, they should beware that the website is able track their browsing activities using cookies. It is known as first party tracking.

A more important concern is: Could an unauthorized and unrelated third party create and access cookies to track a user? The answer is possible. Two well-studied security vulnerabilities, “cross-site scripting” and “cross-site request forgery”, allow third party websites tracking users through a first party website.

HTML5 Web Storage: What is it?

HTML5 Web Storage is introduced to overcome some weaknesses of HTTP Cookie. Comparing with HTTP cookie, it provides user with more control over their data.

HTML5 Web Storage uses a Storage object that represent a list of key-value pairs, much like those implemented in HTTP cookie. The object allows for simple functionality, with methods including SET, GET, REMOVE, and CLEAR methods. Web browser could perform these methods by communicating with this locally stored Storage object. No data is not transferred over HTTP messages.[9]

The main advantages of HTML5 Web Storage compare with HTTP cookie include[10]:

(1) Performance is improved because the frequency of communications between web browser and web server is reduced.

(2) User identifying information are not sent via unsecured HTTP header.

(3) The access of HTML5 Web Storage object is restricted objects to scripts originating at the domain of the top-level document of the browsing context. In the other words, the cookies are not opened to third parties.

HTML5 Web Storage: How trackers use it?

One of the goals of HTML5 Web Storage is to solve the security issues of HTTP cookie. It restricted the access of session information, which prevent unauthorized third party to steal user identifying information by sending request to web browser directly.

However, tracker could still use techniques such as “cross-site scripting” and “cross-site request forgery” to track user’s activities. HTML5 Web Storage is accessible by JavaScript code. Although JavaScript allows websites to perform various functions, it is vulnerable to code injection. Trackers may insert JavaScript code into first party website, get the authentication, then request user identifying information from web browser.

Flash Cookie: What is it?

Using “local shared object” feature in Adobe’ Flash, as known as “Flash Cookie”, is an alternative way to maintain a stateful session connection between web browser and web server. Flash cookies are often share the same values as HTTP cookies, and with below characteristics[11]:

(1) Flash cookies are *long-lived*. They do not have expiration date by default.

(2) Flash cookies’ value could be *stable*. Flash cookies are stored in a different location than HTTP cookies. So that different browsers installed on a given computer access the same persistent Flash cookies.

The long-lived and stable Flash cookies are perfect substitution of “identifying cookie”. Trackers could track users’ browsing activities in multiple sessions using Flash Cookies.

Flash Cookie: How trackers use it?

Trackers can identify users using Flash cookies in the same way as using HTTP cookies. Once a user with unique cookie is connected to the web server, he is identified. Unlike HTTP cookies, tracking cannot be prevented by clearing those cookies stored in browser. Flash cookie could be used for respawning deleted HTTP cookies, and tracker could continue tracking using the respawned cookies with same values.

Comparison of HTTP Cookie, HTML5 Storage and Flash Cookie[12]

	HTTP Cookies	HTML5 Storage	Flash Cookies
Storage	4KB	5Mb by default	100KB by default
Expiration	Session by default	Permanent by default	Permanent by default
Location	In SQL file (Firefox)	In SQL file (Firefox)	Stored outside browser
Access	Only by browser	Only by browser	By multiple browsers on same machine

Cookie is a piece of information stored in users’ machine, to maintain stateful connection between web browser and web server. HTTP Cookie, HTML5 Storage and Flash Cookie are three different methods to store cookies. Comparing with HTTP Cookie,

- *Flash Cookies*: It is long-lived, and can be easily access. It is a perfect choice for “identifying cookie”. The protection on user’s privacy is weak.
- *HTML5 Storage:* It is long-lived, but the access is restricted. It is easy for first party to track users by setting “identifying cookie” in HTML5 Storage. However, it is difficult for unauthorized third parties to get user’s cookies.

There are various techniques adopted by third parties to obtain users’ cookies, then track users. The two well-known techniques, “cross-site scripting” and “cross-site request forgery” will be introduced in next sections.

■ Problems of Stateful Tracking: Getting Cookies by XSS or CSRF

XSS: What is it? How trackers get cookies using XSS?

Cross-Site Scripting ("XSS") is an attack against web applications in which scripting code is injected into the output of an application that is then sent to a user's web browser. In the browser, this scripting code is executed and transfers sensitive data to a third party (i.e. the attacker or tracker).[13] There are two traditional methods of XSS[14]:

- Stored XSS

  Stored XSS is those where malicious code is persistently stored in a resource (e.g. an image of advertisement) managed by the web application, such as a database. When a user requests a dynamic page that is constructed from the contents of this resource (e.g. the host of website put the advertisement image in a webpage), the malicious code is executed. The "identifying cookies" are sent to attacker.

```
Look at this picture!
<img src="image.jpg">
<script>
  document.images[0].src = "http://evilserver/image.jpg?
    stolencookie=" + document.cookie;
</script>
```

  Figure 2: Resource with malicious code[15]. HTTP cookie is sent to tracker if the image is loaded by web browser.

- Reflected XSS

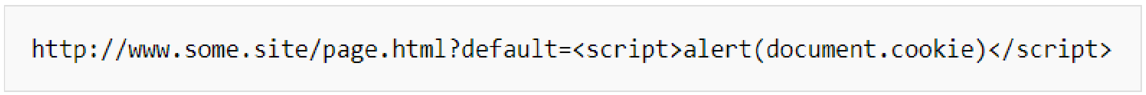
  Reflected XSS is those where malicious code is reflected off the web application, such as in a search result, or any other response that includes some of the input sent to the web application as part of the request. Link with malicious code is delivered to users via another route, such as URL some other website. When a user clicks on the link, the malicious code travels to the vulnerable web site, which reflects the attack back to the user's browser. The browser then executes the code. The "identifying cookies" are sent to attacker. [16]

Stored XSS and Reflected XSS are also known as "server-side XSS". It exploits the vulnerability of server-side code.[17] Trackers pretend that they are trusted web servers, then send malicious code to users for execution.

Another type of XSS, known as "client-side XSS", exploits the vulnerability of client-side code. Browser executes malicious code not because it looks like content comes from trusted websites. It exploits the vulnerability of how browsers interpret a HTTP payload. Malicious code is executed accidentally, where the browser does not verify it is from trusted website or not. The most famous client-side XSS is called DOM-based XSS.[18]

- DOM-based XSS

  In DOM-based XSS, the attacker tries to manipulate the DOM (Document Object Model) of HTML payload. In a typical DOM-based XSS attack, the attacker injects malicious code in strings and sent to user. As some point, the string is converted to code as a result of normal browser activities. When the malicious string is converted to code, a JavaScript DOM is created and executed. The creation of JavaScript is not expected by user's browser, and code is executed accidentally.[19] The "identifying cookies" are sent to attacker.



```
http://www.some.site/page.html?default=<script>alert(document.cookie)</script>
```

Figure 4: An example of DOM-based XSS using URL[20]. JavaScript DOM is created and "alert(document.cookie)" is executed.

A key characteristic of XSS is that it is an attack bypassing authorization. Users usually authorize trusted first party websites to request for cookies. XSS is an attack inserting codes into first party websites, then tricks users to send cookies to third parties without proper authorization.

CSRF: What is it? How trackers get cookies using CSRF?

Cross-Site Request Forgery ("CSRF") is an attack against web applications in which the attacker forces user's web browser to perform an unwanted action on a user trusted website without user's interaction in this action. Attacker pretend to be the user, and attempt to obtain information from web server[21].

The classic CSRF attack, Login CSRF, involves three parties: victim user, trusted site and malicious site. The victim user should be login to the trusted site and simultaneously visiting the malicious site. Since victim user is logging in the trusted website, he would work in a valid session. At the same time, the malicious site provides a malicious link to the victim. When victim mistake clicks this link site, a HTTP request is sent to the trust site with a valid session. Then the trusted site replies user information to victim user. The malicious site will use techniques such as XSS to access those replies from trusted site.

However, there are many limitations for launching CSRF.

- Target trusted website must not have check referrer header.
- Attacker should know the structure of the trusted website to craft a HTTP request.
- User must be logged in trusted website while clicking on the malicious link.

User identifying information, such as cookies and searching history, may be obtained from trusted website if CSRF is successfully launched.

Comparison of XSS and CSRF

In XSS, unauthorized third-party tracker is attempting to obtain identifying information from web browser. Malicious codes are inserted in trusted website, and cookies are sent from web browser to tracker bypassing any authorization.

In CSRF, unauthorized third-party tracker is attempting to obtain identifying information from web server. Malicious codes are included in untrusted website, and cookies are sent from web server because tracker pretends that he is the authenticated user. Then tracker should use techniques such as XSS to obtain those replies from web server.

Preventing browsers to execute malicious code, either inserted in trust website (XSS) or existed in untrusted website (CSRF), remains a challenging task. To protect user's privacy, users may choose another approach: preventing the scripts to connect to unauthorized third party, then send cookies. More details will be discussed in "Ad-blocker" section.

■ Solutions to Stateful Tracking: Preventing Identifying Cookies

User could prevent being tracked by identifying cookies if they delete browsing history manually. Furthermore, most of the web browsers support privacy mode. Cookies would not be saved in browser when user is browsing in this mode.

To prevent the creation of identifying cookies, it is likely to be a policy issue rather than a technical issue. Below two policies are applied in the Internet now[22].

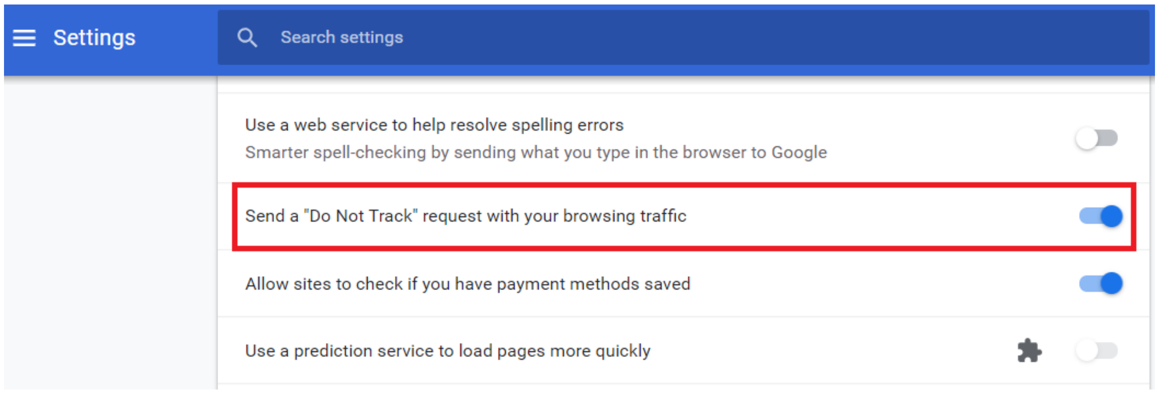
- Opt-Out Cookies and the AdChoices Icon

  Opt-out cookies are non-identifying cookies. User may choose to use opt-out cookies when browsing a website.

  AdChoices is an icon shown on advertisement. It tells users that they are tracked, and advertisements are suggested referring to their browsing records. Users could switch to use opt-out cookies by clicking the AdChoices Icon.

- Do Not Track

  Do Not Track is a standard proposed by W3C, which is a combination of policy and technology. User could enable the Do Not Track preference in web browser to add a *DNT* header in HTTP requests sent. Websites would not track users if they adopt the Do Not Track policy.



Use a web service to help resolve spelling errors
Smarter spell-checking by sending what you type in the browser to Google
Send a "Do Not Track" request with your browsing traffic
Allow sites to check if you have payment methods saved
Use a prediction service to load pages more quickly

Figure 5: Setting to "Do Not Track" in Google Chrome Browser.

Both solutions highly rely on the self-regulation of websites. Reputed first party may strictly follow the policies to protect users' privacy. However, it is not a robust solution to third party web tracking.

■ Solutions to Stateful Tracking: Preventing Execution of XSS and CSRF

XSS: How to prevent it?

XSS is an attack which aims to force users to execute malicious code bypassing authorization. Two approaches are adopted to prevent XSS.

(1) Server: Prevent the insertion of malicious code, i.e. data tainting.

(2) Browser: Prevent the execution of malicious code.

<u>Preventing XSS in Server</u>

Host of websites should prevent trackers to insert malicious code into the webpages. All resources receive from third parties, either persistent or non-persistently, which are later passed to users via the web server should not contain any scripts. The method for filter out malicious code is called “sanitization” or “data tainting”.

Data tainting means that server should ensure that input from untrusted sources cannot be converted to new scripts or modify existing scripts. All resources from third parties are marked as potentially malicious. Any attempt to the input to invokes a subshell, modifies files, directories, or processes will be aborted with an error.[23]

However, data tainting cannot not 100% eliminate the threat of server-side XSS. Nowadays, webpages are dynamic and complicated. It is a difficult task for first parties to guess all possible input from third parties, then design a robust method to test input is safe or not.

Other than the “blacklist” approach, website owner may consider the “whitelist” approach. Web browser should execute the trusted scripts only. All unknown scripted are deemed to be malicious and blocked.

The World Wide Web Consortium (“W3C”) suggested the Content Security Policy[24], which is a guideline for web developers to set policy controlling resources which a particular page can fetch or execute. The XSS problem could be solved at web development level. Malicious resources are not allowed to be converted to code, because they are treated as HTML data under Content Security Policy.

Some researchers suggest that the root cause of XSS vulnerabilities is “the current web application mode violates the principle of code and data separation”[25]. For webpages, data is the HTML content of a page and code is the JavaScript code. In current web application model, HTML data and JavaScript code are mixed in the same channel (i.e. the HTTP response). Attackers are possible to convince a user’s browser to interpret maliciously crafted HTML data as JavaScript code. In year 2013, a group of researchers attempted to use a novel approach to solve XSS problem: separate code and data of a website. A tool was built for or separating code and data in existing websites. All resources from third parties should be treated as data, which is not executable.

Preventing XSS in Browser

Browser should prevent the execution of malicious code. Different browser plug-in are developed to sanitize HTTP responses received from web server. Plug-in filters may:[26]

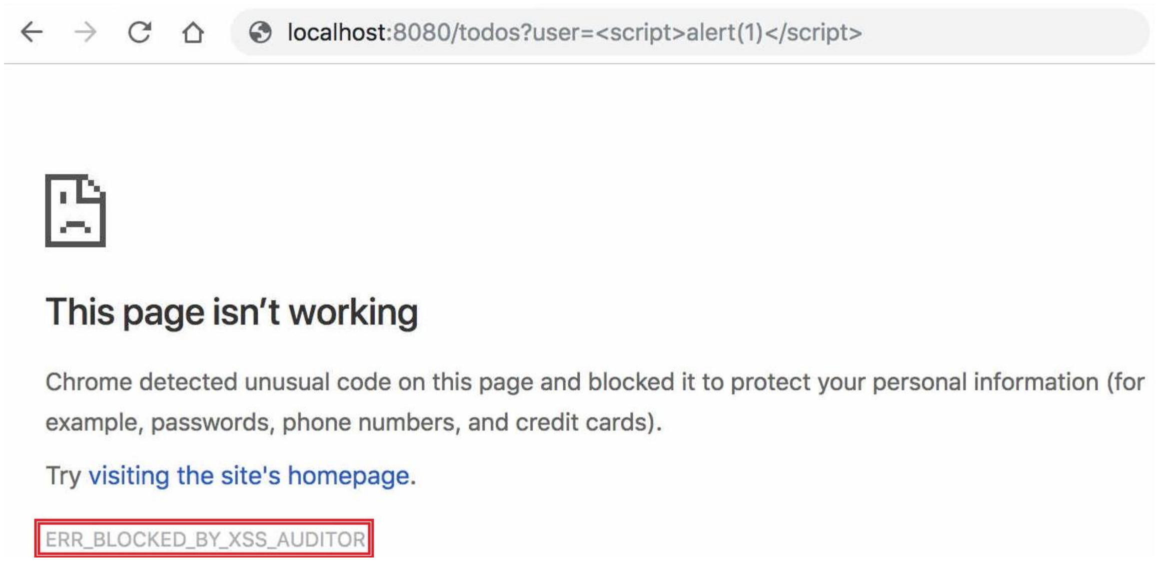
(1) analyse HTTP responses before the responses are processed by the browser; or

(2) analyse HTTP responses after the responses are parsed to HTML, but before executed by the browser.

For (1), XSS filters simulate the HTML parser. HTTP responses are translated to HTML, and the content is analysed. If the HTTP responses is safe, it will be processed by the browser. The main drawback of this approach is the low performance. HTTP responses are processed twice before users could read the webpages.

For (2), XSS filters allows the browser to re-parse HTTP responses to HTML. The filters analyse the scripts in HTTP, then block malicious codes. Browsers could execute scripts only if XSS filters sanitize all malicious scripts. The performance is improved but it is not a robust solution to all types of XSS attack.

The default XSS filter in Google Chrome is “XSS Auditor”[27] which implement (2) only. It monitors all new scripts created in the process of re-parsing HTTP response, and raise alert to user. A commonly-known XSS filter in Firefox is “NoScript”[28], which supports (1) and (2). It targets to defences against all three types of XSS: stored XSS, reflect XSS and DOM-based XSS.


localhost:8080/todos?user=<script>alert(1)</script>

This page isn’t working

Chrome detected unusual code on this page and blocked it to protect your personal information (for example, passwords, phone numbers, and credit cards).

Try visiting the site's homepage.

ERR_BLOCKED_BY_XSS_AUDITOR

Figure 6: “XSS Auditor” in Google Chrome Browser.

CSRF: How to prevent it?

CSRF is an attack which attempt to force web browser to send crafted HTTP request to web server. To prevent CSRF, additional information is included in HTTP header to increase the difficulty of crafting HTTP request. Web server should ignore HTTP requests if their headers are invalid.

Three HTTP header modification approaches are commonly adopted:

(1) Secret Validation Token

(2) The Referer Header

(3) Custom HTTP Header

Researchers found that these approaches could effectively reduce the threat of CSRF, especially for using Custom HTTP Header[29]. However, it leads to other problem: HTTP header are not encrypted by default. This additional information in HTTP header may leak user's browsing records to outsider. Attackers may re-identify users by analysing the HTTP header. Hence, HTTP header should be carefully designed to prevent this security issue.

■ Solutions to Stateful Tracking: Preventing Connecting to Third Party

Preventing the execution of malicious code to send cookies to third party remains a challenging task. Researchers are still working on a robust solution to protect users from XSS and CSRF. However, for preventing stateful web tracking, there is another approach: prohibit the executed code to send any information to third party. The technique is known as “blocking”, which is blocking any network connections with third party trackers.

Ad-blockers: How they work?

Ad-blockers are browser extension to block or hide advertisements. It protects users’ privacy by blocking trackers[30]. The core of Ad-blockers is a filter list, which is a set of syntactic matching rules to block (or allow) the retrieval of a URL by a browser. If ad-blockers are installed and enable, any information exchanges between web browser and listed third party trackers are blocked.

A recent research in 2016 shows that ab-blockers are effective and able to block most connections to tracker web domains[31]. However, even in the best cases, there is still a few percent of tracker domains are not blocked. Three causes are suggested by the research team:

(1) Trackers domains are hide due to the *DNS CNAME* alias. Ad-blocker could filter out tracker domains on list only. Trackers may set up new domains which are not on the filter list, then request web browsers to send cookies to these new domains. This can be solved by regularly update the filter list.

(2) Trackers domains are under first party domain. In case that users are browsing a website under a tracker’s domains, such information exchanges are not blocked.

(3) Some information exchange between web browser and trackers domains are “leaked” and not blocked. The possible reason of this phenomenon is the browser’s extension mechanism are not fully capable with the ad-blockers extensions.

Ad-blockers rely on filter list to protect users from web tracking. Only known third party trackers on the list could be blocked.

■ Future Work

In this survey, stateful web tracking techniques are discussed and introduced. The solutions could be classified in two catalogues:

(1) From policy view: prevent identifying cookies

(2) From technical view: prevent execution of web tracking scripts

Understanding web tracking is a challenging task for the rapidly changing Internet world. Solutions should be continuously evolved to fit the changing world. Future works include:

(1) Continue to understand web tracking by developing tools to scan websites in the Internet.

(2) Continue to improve the design of webpages and browsers by understanding the existing vulnerabilities.

Understand Web Tracking

Researchers develop tools to measure the most recent web tracking activities in the Internet, such as "*FourthParty*" in year 2012 and "*OpenWPM*" in year 2016. The suspicious scripts inserted in popular websites are scanned and recorded for further analysis. These tools rely on researcher's manual input to identify the suspicious scripts. It is also possible to return a false positive result. In future, more efficient tools could be built by applying techniques such as machine learning. If success, it could greatly improve the efficiency and accuracy of browser privacy tools[32].

Improve Design of Webpages and Web Browsers

Researchers continuously study the vulnerabilities in websites. They propose methods to detect and fix those vulnerabilities. For instance in year 2019, researchers systematically study a new type of XSS called "Persistent Client-Side Cross-Site Scripting"[33]. For this attack, malicious codes are inserted into HTML5 Local Storage. If the browser accesses the Local Storage, the codes are retrieved and executed.

Researchers also proposed new standards for webpages design, such as Content Security Policy, to eliminate any chances of inserting malicious codes in webpages. The standards are continuously reviewed and updated. For example, in year 2016 researchers from Google concluded that the whitelist approach of CSP is not sufficient for preventing XSS[34]. They suggest using a nonce-based approach. Webpages could include nonce in scripts to prevent any code injection.

■ Related Work

Several well-known surveys are conducted to summarize existing web tracking issues. They target to conclude the existing problems, and provides a basis for researchers and regulators to further discuss those problems.

Trackers and Tracking Techniques. Adam Lerner at el.[35] study the changes of trackers and tracking techniques from year 1996 to year 2016. For trackers, it is noted that *google-analytics.com* is found in one-third of websites since 2010s. For tracking techniques, the use of HTML5 Local Storage grows rapidly in year 2009 which suggest that tracking defences against it should be considered more.

Identifying Cookies. Mike D. Ayenson at el.[36] study different types of cookies which identify users' information. Steven Englehardt[37] further study how pervasive is web tracking using cookies.

Cross-Site Scripting. Gupta, S., and Gupta, B. B.[38] study the changes of XSS and its defence mechanism. Tools are designed to eliminate the threads of XSS. The weaknesses of those tools, which should be overcome in future, are analysed and summarized.

In this survey, a brief introduction of each topics is given. It is aimed to provide reader a big picture of stateful web tracking. The topics covers are wider but shallower. Other difference with existing surveys is the relationship between each topic, such as web tracking and web attacks, are discussed.

■ Conclusion

Stateful web tracking means a tracker re-identify a user by cookies, even after a browsing session is ended. It involves two parts:

- How do first party track users: Craft identifying cookies
- How do third party track users: Get cookies using XSS or CSRF

First party tracking involves identifying cookies. There are three types of cookies: HTTP cookies, Flash cookies and HTML5 Local Storage. Either type of cookies could be crafted to be long live and unique, which allows first party to track users in multiple browsing sessions.

To defence against tracking using cookies, policies such as “Opt-out Cookies” and “Do Not Track” are imposed. Website promises not to creating identifying cookies if user wishes. However, these solutions highly rely on the self-regulation of website owners.

Third party tracking may involve web attacks, such as XSS and CSRF. Some web tracking activities are authorized, which user’s information are passed to third party via first party. Some web tracking activities are unauthorized, which tracker launch web attacks to steal the identifying cookies from user.

To defence against stealing cookies, two approaches could be adopted: prevent the execution of malicious codes or prevent the network connection with tracker domains.

Malicious codes are executed in user’s browser to steal cookies because trackers launch attacks such as XSS or CSRF. Various kinds of defences mechanism are invented to protect users against the attacks; however, a robust solution is still not found.

Network connection with tracker’s domains could be stopped by ad-blockers. The cookies cannot be sent to tracker’s domains if ad-blockers work properly. However, the core of ad-blockers is filter list which requires browsers having knowledges on trackers. Only well-known trackers could be blocked by ad-blockers.

References

[1] Lerner, A., Simpson, A. K., Kohno, T., & Roesner, F. (2016). Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*.

[2] Englehardt, S., & Narayanan, A. (2016, October). Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1388-1401). ACM.

[3] Google Chrome Privacy Notice. (2019, March). Retrieved from *https://www.google.com/chrome/privacy/*

[4] Mayer, J. R., & Mitchell, J. C. (2012, May). Third-party web tracking: Policy and technology. In *2012 IEEE Symposium on Security and Privacy* (pp. 413-427). IEEE.

[5] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *RFC 2616: Hypertext transfer protocol–HTTP/1.1* (No. RFC 2616).

[6] Kristol, D. M. (2001). HTTP Cookies: Standards, privacy, and politics. *ACM Transactions on Internet Technology (TOIT)*, *1*(2), 151-198.

[7] Barth, A. (2011). *HTTP state management mechanism* (No. RFC 6265).

[8] Englehardt, S., Reisman, D., Eubank, C., Zimmerman, P., Mayer, J., Narayanan, A., & Felten, E. W. (2015, May). Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web* (pp. 289-299). International World Wide Web Conferences Steering Committee.

[9] West, W., & Pulimood, S. M. (2012). Analysis of privacy and security in HTML5 web storage. *Journal of Computing Sciences in Colleges*, *27*(3), 80-87.

[10] Web Storage (Second Edition): W3C Recommendation. (2016, April). Retrieved from *https://www.w3.org/TR/webstorage/*

[11] Soltani, A., Canty, S., Mayo, Q., Thomas, L., & Hoofnagle, C. J. (2010, March). Flash cookies and privacy. In *2010 AAAI Spring Symposium Series*.

[12] Ayenson, M. D., Wambach, D. J., Soltani, A., Good, N., & Hoofnagle, C. J. (2011). Flash cookies and privacy II: Now with HTML5 and ETag respawning. *Available at SSRN 1898390*.

[13] Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G. (2007, February). Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *NDSS* (Vol. 2007, p. 12).

[14] Cross-site Scripting (XSS). (2018, May). Retrieved from *https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)*.

[15] Same as [13].

[16] Pauli, J. (2013). *The basics of web hacking: tools and techniques to attack the Web*. Elsevier.

[17] Doupé, A., Cui, W., Jakubowski, M. H., Peinado, M., Kruegel, C., & Vigna, G. (2013, November). deDacota: toward preventing server-side XSS via automatic code and data

separation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 1205-1216). ACM.

[18] Klein, A. (2005). DOM based cross site scripting or XSS of the third kind. *http://www.webappsec.org/projects/articles/071105.shtml*.

[19] Lekies, S., Stock, B., & Johns, M. (2013, November). 25 million flows later: Large-scale detection of DOM-based XSS. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 1193-1204). ACM.

[20] DOM Based XSS. (2015, June). Retrieved from *https://www.owasp.org/index.php/DOM_Based_XSS*.

[21] Siddiqui, M. S., & Verma, D. (2011, May). Cross site request forgery: A common web application weakness. In *2011 IEEE 3rd International Conference on Communication Software and Networks* (pp. 538-543). IEEE.

[22] Same as [4].

[23] Same as [13].

[24] Content Security Policy Level 3. (2018, October). Retrieved from *https://www.w3.org/TR/CSP3/*.

[25] Doupé, A., Cui, W., Jakubowski, M. H., Peinado, M., Kruegel, C., & Vigna, G. (2013, November). deDacota: toward preventing server-side XSS via automatic code and data separation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 1205-1216). ACM.

[26] Bates, D., Barth, A., & Jackson, C. (2010, April). Regular expressions considered harmful in client-side XSS filters. In *Proceedings of the 19th international conference on World wide web* (pp. 91-100). ACM.

[27] XSS Auditor. Retrieved from *https://www.chromium.org/developers/design-documents/xss-auditor*.

[28] NoScript Security Suite. Retrieved from *https://addons.mozilla.org/en-US/firefox/addon/noscript/*.

[29] Barth, A., Jackson, C., & Mitchell, J. C. (2008, October). Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security* (pp. 75-88). ACM.

[30] Pujol, E., Hohlfeld, O., & Feldmann, A. (2015, October). Annoyed users: Ads and ad-block usage in the wild. In *Proceedings of the 2015 Internet Measurement Conference* (pp. 93-106). ACM.

[31] Wills, C. E., & Uzunoglu, D. C. (2016, October). What ad blockers are (and are not) doing. In *2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)* (pp. 72-77). IEEE.

[32] Same as [2].

[33] Steffens, M., Rossow, C., Johns, M., & Stock, B. (2019). Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild. In *2019 Network and Distributed System Security Symposium*.

[34] Weichselbaum, L., Spagnuolo, M., Lekies, S., & Janc, A. (2016, October). CSP is dead, long live CSP! On the insecurity of whitelists and the future of content security policy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1376-1387). ACM.

[35] Same as [1].

[36] Same as [12].

[37] Same as [8].

[38] Gupta, S., & Gupta, B. B. (2017). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, *8*(1), 512-530.