Protection of Web User’s Privacy by Securing Browser from Web Privacy Attacks

Sanket Baviskar and Dr. P. Santhi Thilagam
Department Of Computer Science and Engineering, National Institute Of Technology, Surathkal.

Abstract

The Internet and World Wide Web being an essential part of everyone’s life, the users have become more conscious about their privacy. The user’s web privacy unfortunately, is not focused much by the current browsers. This paper focuses on the attacks which compromises user’s anonymity, grab user’s private data. It highlights the privacy attacks such as history sniffing, web beacons, browser fingerprinting. We focus on the protection of user’s information which is transferred over the network to the web server without the users consent rather than stealing attempts which fools the user and trap them to install malicious software which perform the theft, using social engineering techniques. A generic algorithm has been proposed in this paper which will stop the attacker from violating privacy. We have proposed a secure way of interaction between browser objects through browser API’s and the JavaScript. We developed an add-on which implements the proposed solution.

Index Terms—web privacy, history sniffing, browser fingerprinting, web beacon.

1. INTRODUCTION

Individual privacy is an important dimension of human life. The need for privacy is almost as old as the human species. Definitions of privacy vary according to context, culture, and environment. The privacy of a person can be defined as “the right to be let alone.”[3] In other way, “the desire of people to choose freely under what circumstances and to what extent they will expose themselves, their attitude and their behavior, their personal information to others.” With the wide use of computers and the emergence of the Web, privacy has also become a digital problem. In particular, with the revolution in web, privacy has become a serious problem which has many upcoming challenges different from the previous web era. In the context of web, “right of the user to have control over the data collected by websites and reveal the identity based on his/her wish.” commonly referred to as Web privacy. In general, the phrase Web privacy refers to the right of Web users to conceal their personal information and have some degree of control over the use of any personal information disclosed to others.

Web user’s private information can be categorized as personal, digital or electronic data[3]. Data such as person’s name, mailing and email addresses, phone number, financial information and health information, marital status, which can be referred as personal data. Data related to web user’s activity like their site visits, frequency and duration of their site visits, online shopping patterns can be referred as digital behavior of the web user. Data such as web user’s electronic message posting to electronic boards and votes submitted to online polls. Personal data and data submitted through communication, for which the user is aware of because it needs the user interaction to submit the data and the user may be aware of the risk associated with that data. But in case of digital behavior, user is not aware of the theft as the information is stolen without the user’s concern, without user’s knowledge. In this paper, private information refers to any information transferred over the network, captured by the interaction between the page script code and browser objects which can help to create identity of the user and reveal certain secrets about the user’s browsing pattern.

1.1. Failure of current privacy policies

The current approaches followed on the web for the web user’s privacy protection fails and are not so effective because—

- User has to trust the privacy policies of web sites which may not be safe, because web sites may track the user’s and sell their information to others, though they mention not to do so in their privacy policies.
- Many web sites may not strictly follow the privacy policies mentioned by them.
- Current measures need the user to understand the technical issues such as deleting cookies, deleting history enabling/disabling JavaScript in order to have better control over the user’s information flow over the Internet.
It is very difficult to identify which site gathers what through user’s browser unless and until the user analyze the script code and find calls made to objects through browser API’s.

1.2. Risks involved in privacy theft

The list of Web locations visited by the user often conveys detailed information about the user’s family, financial or health situation. This information may be sold to the third parties by the visited web sites. Customers using online banking service can be attacked by the method of Phishing when some websites can grab the data from the user’s browser related to banking sites. Phishing—a fraudulent attempt made to steal user’s personal information. It allows E-commerce companies to adjust ads or prices—for instance, if the site knows you’ve just come from a competitor that is offering a lower price. The advertising companies can target the users with their advertisements, when they have enough information about browsing interest of the user.

1.3. Motivation

There are many ways possible by which the user’s privacy can be compromised over the network and private data can be stolen. Private information such as browsing history, saved passwords, bookmarks, cookies, etc. are stored in the local file systems on the user’s system. To steal this kind of information by induceing the user to visit the malicious site on the Internet, there are following possible ways:

First is to execute some code in user’s system that will steal the private data. It can be done by installing an add-on in the user browser. Because add-on has all the privileges to access the local file system. But installing any add-on requires user’s permission. Other way is through applets. When any applet is downloaded to the user’s browser, they are run in java sandboxes[4] which will restrict them to have access to local files. Another approach is by inserting a virus, rootkits, etc. into the user’s system through Internet. They can transfer the information to the web server secretly.

To successfully execute above mentioned approaches, either the user has to interact, as in case of the add-on installation and malicious softwares or bypass the browser’s sandboxing mechanism to execute programs such as applets. By using some social engineering techniques the user can be fooled to download malicious codes into the system.

In this above approaches mentioned, there is no guaranteed success to the attacker, as they depend either on user’s interaction or face privilege restrictions. At one time these type of intrusions can be taken care by the firewalls and anti-viruses, etc. Also user’s can be educated about safe downloading methods.

But there are some methods by which neither the user get to know about the information theft nor the current browser has any way of preventing the theft. Those attacks do not need user’s permission to get executed and steal some data from the browser without user’s consent. This fact became the main motivation for this paper to have a strong preventing mechanism against these type of thefts generally referred as web privacy attacks. There are certain issues such as if the user wants to preserve his/her anonymity on the Internet or if the user doesn’t want his/her browsing history to be compromised, that need to be addressed.

2. WEB PRIVACY ATTACKS

2.1. History sniffing¹

It is a browser’s rendering engine exploit. The attacker can determine whether the user has visited a particular site or not based on the rendering method used by the browser for differentiating between already visited and newly visited site[1]. Suppose a malicious site (say, abc.com) developer wants to learn whether a user has visited a specific URL (say, xyz.com) or not, merely by inducing the user to visit abc.com, using history sniffing technique. Hence the attacker appends a link in the web page to the target URL (here xyz.com) and hides it from displaying to the user by setting the visibility of the links as hidden. Attacker uses the browser's DOM interface to inspect how the link is displayed. Attacker retrieves the color of the link( xyz.com). Then based on the color of the link, it’s concluded that whether xyz.com is visited by the user or not. Attacker can do it for thousands of users without their consent. Hence, the attacker uses the fact that browser display links differently depending on whether or not their target has been visited. If the link is displayed as a visited link, the target URL is in the user's history. And if not, then it is newly visited.

2.1.1 Threats due to history leakage

History sniffing can enable several kinds of privacy violations on the part of unscrupulous sites. For example [1], history sniffing can be used by phishers to better target users to phishing sites, Phishing is a fraudulent attempt to steal user’s personal information. By the investigators (e.g., insurance companies) to check if a user has been reading on some topic (e.g., a specific medical condition), and by the advertisers to build more complete user profiles, or by cooperators sites to track the users even when user has disabled cookies.

2.2. Browser fingerprinting [6]

Generally, people assume that they can prevent a website from identifying them by disabling cookies on their web browser. But this is not true always. When user

¹ history sniffing is possible provided that the user has not cleared the browsing history.
visits a website, user is allowing that site to access a lot of
information about user's computer's configuration and
browser details. Combined all the information, can create
a fingerprint — a signature that could be used to identify
you and your computer.
As an example, a person can't be identified provided only
its gender or date of birth or ZIP code but can be identified
when this information is combined together to create
personal identity.
Similarly in case of a browser, information such as
browser details[7], User Agent, HTTP_ACCEPT header,
plugins installed in the browser, time zone of the user,
fonts installed, screen details, can form a fingerprint and
then can be used to identify the user's browser uniquely.
So by any means the user's browser can be identified by
the web sites the user visit even though the cookies or
flash cookies are deleted. It compromises the user's
anonymity.

2.3. Web bug

Web bugs[2] are small graphic images placed on web
pages or in an email message to facilitate third-party
tracking of users and collection of statistics. Usually, a
web bug is a transparent image or an image in the color of
the background of your web page. It is typically 1*1
pixels in size. They are hosted by third-party sites.

2.3.1. In a web page

Usually a Web Bug is aggregated to a Web page
through images with some special properties. These
images are hidden from the user and hosted by third-party
sites. When a user visits a Web site that links to such
images, his/her browser sends a requests for them, and so
user's IP is sent to the third-party host. Depending on the
pages linking these images, user's IP path would be
tracked by visited sites. Web Bugs work with cookies. A
cookie is a textual information stored in user computers to
identify Web site visitors. When a user first requests a site
with Web Bugs, cookies are recorded in his/her computer.
Thereby following requests to the Web site will be enough
to identify the user. In fact, the most common way are
images, represented by HTML code, but any HTML
element able to reference a third-party site (e.g. script,
iframe, div ) could also be used.

2.3.2. In an e-mail

Web bugs are frequently used in spamming (sending
unsolicited commercial e-mail) as a way of "pinging" to
find which spam recipients open (and presumably read)
before deleting it. It is also used to confirm whether the
particular email address is working or not. When the email
containing the web bug is forwarded to other email
address, the recipient opens that mail and the request for
that web bug is forwarded to the hosting site. It can be
concluded that the email is forwarded.

2.4. Commonality in privacy attacks

Privacy attacks are nothing but stealing the browser’s
information and then transferring it to the network. It can
be observed that in order to steal the user’s private data—
Stored by the browser— by the attacking site, the script
code has to interact with the browser APIs to access browser
objects and object properties. In case of history
sniffing attack, the only possible way to access the color
of the link is by accessing the style property of the link
object.

As scripts do not have privilege to access local file system
all the data such as cache data, history data can’t be
accessed directly.

There are two possible ways to do so:

First is to execute some code in user’s system that will
steal the private data. It can be done by adding add-on in
the user browser. But installing any add-on requires user
permission. So this possible way may not work always.
Similarly even if any applet is downloaded to the user’s
browser when it navigate to the attacking site, as applets
are run in java sandbox which will restrict them to have
access to local files. Inserting a virus through Internet
cannot be considered as the browser flaw at all. They can
be taken care by the firewalls and anti-viruses.

In the above methods mentioned, it can be observed that
there is at least one chance for the user to accept the add-
on or applet. But what if the user is completely unknown
about the theft. This can be done by script code execution
in the web user browser. The script gets executed and
gathers certain information. Based on that information
some conclusions are obtained. It may be about the users
browsing behavior, browsing history, user identity.

3. PROPOSED SOLUTION

In order to make the user aware of the information
Theft, it can be intimated prior to information flow out of
the web browser. But instead of intimation and interaction
of the user to stop the information flow, it is better to have
security at the browser level only. So in this paper a
generic approach is proposed which will filter the access
to the browser’s objects through the script code. As like in
the history sniffing[1] attack, if the attacker is not able to
access the style property of the link inserted or if null
value is returned for the link style, then it can’t be
concluded if the site is visited before or not. In case of
browser fingerprinting, when the attacking script code is
executed in the user’s browser, it can capture all the
details about the system configuration such as plugin
details, whether cookies are enabled or not?, time zone of
the user. Flow of this information out of the browser is not
under user’s control. What if the user wants to preserve its
anonymity on the Internet? What if the user don’t want its
browsing history to be revealed? To fulfill the user’s
request to open a particular site in the browser, it is not
necessary to transfer any of the information mentioned
above over the Internet to the web server. User should be safe enough and assured by the browsers that any of the information transferred out of the browser while browsing on the Internet, don’t have any threat of identifying the user. That information should not be any data personal to the user such as browsing history, saved passwords, etc. What if all the unnecessary information transferred over the net through browser is filtered before the transfer, sending only the safe information. Safe information in the sense, information which possess less threat to users privacy. It would definitely have the control over privacy leak. The algorithm proposed in this paper does the above mentioned thing which is implemented by developing an add-on for the Firefox browser.

3.1. Algorithm

- User type an URL in the browser address bar and press enter to send the request to the web server for a particular web site.
- The response from the web server contains the requested web page content.
- Before the web page gets rendered into the browser and displayed to the user, stop it.
- Parse the JavaScript page to extract all the calls made to the browser objects and the object properties.
- Prepare a black list containing the object and its respective property which should not be accessible by the JavaScript, which can leak certain information which will help the attacker to create identity of the user or get some personal data.
- If the objects and properties extracted from the page source, contains any object with its property which match with that in the black list prepared, block the access by returning null value or false value for object property accessed.

3.1.1. Web bug detection algorithm

- Extract all the image tags, and extract their sources.
- Compare the source with the domain name of the URL the user entered.
- If there is a mismatch in the domain name, add the source of the image into the list of possible web bug threats.
- Replace the source of the image tag with blank, so that when the web page is rendered into the browser, the request will not go to the web server where the web bugs are hosted.
- Display the entire list of web bugs to the user and give privilege to the user allowing or stopping him/her to display the images depending on which advertisement the user is interested in.
- As a awareness measure, the user can be notified to delete the cookies after he/she allows for a particular image to be displayed in his/her browser, so that the image hosting server which will add cookies in the user’s browser, will not be able to track user’s browser.
- Rewrite the JavaScript by writing the false values or null values as return value for the blocked objects and web bugs.
- Render the web page with the new JavaScript code and display it to the user.

**Figure 1.** shows the proposed System architecture. The middle rounded rectangle shows the functionality of the add-on i.e. all the steps are followed by the add-on.

As a part of the implementation and to prove the correctness of the method we have developed one add-on which can implement the above mentioned algorithm and can prevent history sniffing, browser fingerprinting and detect possible threats of web bugs. We designed one attacking web page named music_mania.com which performs history sniffing, collect information for the browser fingerprinting and has web bugs embedded. One table is created which displays all the information collected in a particular session from user’s browser.

---

2 As 90% of the web sites are designed using JavaScript, in this paper it is assumed that the web pages are designed using JavaScript only and algorithm is designed accordingly.
4.1. Working of the add-on

4.1.1. Extraction of object calls:
All the calls made by the script code to browser objects are extracted using the regular expressions. Regular expression plays an important role in obtaining the set of all browser objects in the script obtained, that needs to be matched with the black lists prepared.
Regular expressions:
As regular expression works with the string, they can be applied on the response retrieved from the web server. Using these regular expressions first all the occurrences of the browser objects is obtained and a set is formed which in the next steps need to be compared with the black list we prepare which contains objects and their properties. We also extracted all the image tags present in the script code to check for possible web bug threats.

4.1.2. Preparation of the black lists:
We have prepared two black lists:
One is object list:
Which contains the objects whose certain properties or methods when accessed through the script code, possess a threat of revealing certain data which may be private to the web user, thus compromising user’s web privacy. **Figure 2**, shows document object model of a browser in the form of DOM tree. It shows the root object, its child object and their properties.
The black list we prepared, contains those objects, which holds certain properties which when accessed have a possible threat of web privacy violation. If we consider possible way to do so is by accessing the style property of the link object. So this fact adds the link object—whose parent is document object—in the black list of objects. Similar fact is applied for all the object those possess similar kind of threat.
The second list we prepared is of object properties:
As mentioned above, in case of history sniffing attack, the **color property** of the link object has a threat of web privacy violation so this fact adds the method **getPropertyValue** in the black list of object properties.
Content of these black lists is entirely dependent on the intelligence applied in using the object and properties in order to reveal the private data. It is also dependent on how a particular piece of information can be used in compromising the user’s privacy. Important aspect to be considered while generating such black lists is, it should not contain any object or the property which will block the basic functionality of the browser or it should not affect the response, user gets when it sends the request to a web server. That means user should get whatever he/she wants and at the same time the web site should not be stopped from conveying its purpose. For example it happens in case of history sniffing attack, that the links of attacker’s interest are hidden in the web page from displaying it to user. As the fact of hiding the links in the web page also indicates the threat of privacy violation, hence the visibility property of the link object may be blocked, by giving the default value as visible or null value every time it is accessed by the script code. But it may happen that the web developer needs to hide some set of links in the web page designed, in order to have the effective functionality of the web site, rather not to collect the information which will compromise user’s privacy. The designer may want to display a certain set of websites only when the user completes some action. In this case the designer will face certain problem in designing the web site if certain functionality is disabled.
So to avoid such problems, selection of objects and their properties should be done in such a way that it has highest possibility of revealing the private data.
The purpose of making two black lists is to have minimal number of comparisons. Instead of creating the single list which contains only one entry for the object name and its property, it is better to have first the list of objects used in the current web page code, which are blacklisted and then go for checking blacklist properties.

4.1.3. Checking:
It is a two step procedure:
Step 1:
Object validation:
This step performs the task of finding matching object in the object list we obtained in the extraction step with the one we prepared as a black list. This step is needed to eliminate the other unnecessary objects those are not to be considered in the next validation step. So it avoids
comparing each and every object and their properties with the black lists. It gives the object needed for the property validation step.

Step 2:
Property validation:
This step is executed only for the properties of objects, obtained in object validation phase. When a particular blacklisted object is considered, again with the help of regular expression, all the possible occurrences of that object with its respective property accessed, is found out. If any of the properties we obtained after parsing is present in the black list of properties we have, it will be blocked.

4.1.4. Blocking:
This step blocks all the accesses made by the JavaScript code to the browser objects and their properties if they are present in the blacklists we prepared. It does this by giving the false values or the null value for those accesses. Consider the example of browser fingerprinting, where the attacker collects the list of the installed browser plugins by using `navigator.plugins` object. It does this by accessing the description property of the each and every plugin item in the plugins list. So according to the proposed algorithm we return the null description for each of the plugin item. For the Boolean values we have taken special care and it will always return false values irrespective of the original values.

For the web bugs detection, according to the web bug detection algorithm mentioned above, if there is any possible threat of web bugs, we blocked their rendering by setting blank or null source for `<img>` element.

4.1.5. Rendering:
Now we have got newly modified and a secure JavaScript ready to be executed in our browser. so this new script code is fed to the browser’s rendering engine. The script gets executed by the browsers and then displayed to the user as the result of his/her request to the web server.

5. RESULTS
Following Figure 3 shows the test bed we created. It is a sample web site named music_mania.com performing the history sniffing, browser fingerprinting and having web bugs inserted in it, in the form of 1x1 pixel images. Before add-on installation the web page looks like as shown in the Figure 3.

The table shows the information[6] collected from the browser such as user agent, browser version details, plugins installed, screen resolution, etc. The add-on implements all the above mentioned steps and finally mitigates the attacks we considered.

After installing the add-on, web page looks as shown in Figure 4.
The list of possible web bugs is displayed in the user interface provided by the add-on with the URL of the web page containing the bug.

5.1. Issues and assumptions:
Any information that is blocked may be needed to fulfill the user’s web request. e.g., information about the version of the browser. It may happen that some websites work for particular browser version only. In such case if the particular information is needed by the website to know whether the browser is having the required version or not, it can display the notification about that required information so as to display the required web page.

There is an user interface provided by the add-on which allows the user to add or delete the objects and the specific property of that object. So the required information can be transferred or blocked with the user’s permission only. The list of possible web bug threats displayed in the user interface provided, allows the user to load images, which are considered as possible web bug threats, by the user’s permission, like in case of third party advertisements. The goal achieved by providing the GUI is, that the user is aware about the flow of the information out of the browser. User can add or delete the objects and their properties which are suspicious according to his/her view even if they are not focused in the add-on implementation while preparing the black lists.

There are certain assumptions to be made to perform the web privacy attacks. Like in case of history sniffing attack it has to be assumed that the user has not deleted the browsing history, because if it is deleted, then the browser will display the link as unvisited though it is visited before, link in which the attacker is interested to know whether visited or not. Also if the user has changed the default color of visited and non-visited links, history sniffing will not be possible.

When the web page with the web bug is rendered by the browser, the request goes to the third party server where the image is hosted. Third party server can then track the user by inserting the cookies in the web page. If the user deletes the cookies by the third-party server, it will not be able to track the user unless again a request is sent to the web bug hosting server. It is not necessary that web bugs are implemented by inserting the images only. Modern web bugs also use the HTML IFrame, style, script, input link, embed, object, and other tags to track usage. So they can also be mitigated using the similar method proposed.

6. CONCLUSION
The aim of this paper is to address the problem of information flow out of the browser without user’s concern, tracking without the knowledge of the user. The proposed solution will be more effective when it is followed by the current browsers as a part of their architecture. The user will be aware about each and every piece of information moving out of the browser over the Internet. Similar method can be applied for the data transferred over the network using methods such as GET and POST. There may be certain hidden things going on in the browser when user visits the web site, while the web page is rendered, some data may get transferred over the network like in case of history sniffing, the data about the link colors, or in case of browser fingerprinting, system configuration information. There can be lot of ways possible to perform such thefts, each and every attempt can’t be generalized unless the attempt is performed. But if the user gets the knowledge about the risks involved, at least user can avoid these kind of thefts. They can avoid visit to malicious web sites, deleting the cookies received from the particular site. By implementing the proposed method for browser security, the browser will be secure enough on the initial delivery itself, to the user. If the user wishes to have high level of security implementation, he/she can do so by interacting with the browser. The user will have the privilege to set the browser functionalities according to their needs. They will make their browser suitable according to their needs. So ultimately browser will work according to user’s wish and not the user according to browser.

7. REFERENCES