Studying the Privacy Issues of the Incorrect Use of the Feature Policy

Beliz Kaleli
Boston University
bkaleli@bu.edu

Manuel Egele
Boston University
megele@bu.edu

Gianluca Stringhini
Boston University
gian@bu.edu

Abstract—In addition to rendering HTML and providing Web access, Web browsers offer auxiliary features (e.g., camera, geolocation, microphone etc.) that can be used while browsing. Some of these features access sensitive information such as camera image, microphone, and location data. The common behavior of modern browsers when a website tries to access one of these features is to prompt a permission dialog for the user to allow or deny this request. The user’s response may be recorded by the Web browser and remembered for that website. However, websites may include third party resources in the form of HTML iframes. Recently, the W3C introduced the Feature Policy to restrict which origins contained in a website get access to which features. To prevent the leakage of sensitive information, both Web developers and browser developers have to handle feature usage carefully. Web developers need to ensure that permissions are only given to trustworthy websites and Web browser developers have to implement the Feature Policy correctly. A wrong implementation from either one of these parties could cause privacy leaks, such as an untrusted iframe getting access to the user’s webcam.

In this paper, we present a study of nine Web browsers and their behavior regarding feature usage in websites, and show that five of them (Chromium based browsers) are open to social engineering attacks that may lead to sensitive information disclosure to untrusted parties. We identify the root causes of this problem as two-fold: Web browsers prompting misleading dialogs when asking for permissions and the wrong use of Feature Policy by websites.

I. INTRODUCTION

As technology evolves, new requirements emerge for websites and applications to serve users. While location based services have long enjoyed popularity among mobile applications (e.g., [1], [2], [3], [4]), commodity browsers started to support similar APIs and hence allow users to take advantage of the location based functionalities on their Web browser. Similarly, since most social media apps/websites such as Facebook [5] require camera and microphone access for some functionalities (e.g., video-calling), browsers also added support for those features.

To ensure the privacy of user-data, the W3C recently proposed the so-called Feature Policy [6] HTTP response header, which allows webmasters to precisely control which parts of a website can access what features, pending user-approval. The Feature Policy is analogous to the Same Origin Policy [7], in the sense that the website developer and the browser developer both need to work together to achieve the desired protections.

In this paper, we show that the use of Web browser features may lead to sensitive data leakage if not handled carefully. Specifically, embedded iframes in a website can also request a feature access through the main website. The sources of these iframes may be untrusted third-parties. The responsibility of the Web browser is to inform users of feature requests and what it means to “allow/deny” those requests, whereas the website developer should limit feature access to third parties by using Feature Policy correctly. If the chosen Feature Policy is overly-permissive, sensitive user data may also be accessed by untrusted iframe origins.

Previous studies focused on the adoption of response headers and their security and privacy implications [8], [9], [10], [11], [12], [13]. In other work [14], [15], researchers proposed sandboxing scripts as a countermeasure to data leakage. To the best of our knowledge, we are the first to study the adoption of the Feature Policy in the wild, and to look at the risks arising from its incorrect implementation by Web browsers and websites.

To study the extent of the problem, we developed three test cases, applied them to 9 different Web browsers, and observed their behavior. We find that 5 out of 9 Web browsers (Chromium based browsers) are open to social engineering attacks that can result in a third party to access sensitive information from untrusted iframes without the user’s knowledge. We also note that this partial-vulnerability is known by the Chromium developers and was a design choice in response to publicly unavailable surveys.

In summary, the contributions of this paper are as follows:

• We present potential security issues arising from an incorrect deployment of the Feature Policy, and develop three use cases that allow us to identify vulnerable Web browsers and websites.

• We evaluate 9 Web browsers, finding that 5 of the browsers (including Google Chrome) are open to social engineering attacks which may lead to sensitive data leakage.
II. BACKGROUND

In this section, we provide the pertinent background information that the remainder of the paper relies upon. First, we introduce browser features and how permissions can be granted to Web origins. Then, we present the recently proposed Feature Policy, which regulates the feature usage by Web browsers. Finally, we present our threat model of how a malicious party could obtain sensitive information from a victim with an example scenario.

A. Browser Features

Most modern Web browsers offer several features such as geolocation, camera, and microphone, to enable several functionalities on websites (e.g., video-calling, geolocation). To use these features, websites ask the user for permission to access every feature they need. When a website tries to access a feature, the Web browser prompts a permission dialog box to the user. The purpose of this dialog box is to inform the user of the access request, and it typically contains the name of the feature and the origin of the website that requested it, together with “allow” and “block” options. The browser may then record the response to that prompt, which causes a permanent permission, or may alternatively not record the user’s decision and prompt the permission request every time.

Websites may have embedded iframes in them. These iframes may also need to use browser features. In that case the embedded iframes will contain a script in which they ask the browser for permission to access one or more features. Depending on the Feature Policy configuration, the main website could prevent iframes from requesting permissions for features, as discussed next.

B. Feature Policy

Website developers may want to control access to certain browser features within their websites as a security or performance precaution. The Feature Policy addresses this issue by offering the ability to selectively enable/disable the access to the browser features in its own frame, and in content within any iframe elements in the document. The HTTP Feature Policy header is used with a directive and an allowlist for that directive. The directive is the feature to be restricted and may take the values in Table I.

The allowlist can take different values. By specifying the “none” keyword for the list, the specified features will be disabled for all browsing contexts, regardless of their origin. The “self” keyword will disable the use of the specified feature within all browsing contexts except for the main website’s own origin. If a developer wants to enable the feature for their own website or a third-party websites, they will specify the URI of that website as the value of the allowlist. The “self” keyword and specific URLs can be used together. The wildcard option allows the feature to be accessed in the document, and all nested browsing contexts regardless of their origin. For example, the following configuration will allow the camera feature to be used by both the main website and all nested iframes while blocking the microphone usage in all browsing contexts: “Feature-Policy: camera *; microphone ’none’.” If the website contains iframes, the wildcard configuration can be vulnerable since third-party contexts will also be able to access the camera feature. As a result, that third-party will have access to the user’s camera and hence record her and steal sensitive data.

C. Threat Model

Browser features can be requested by both benign and malicious websites. An educated user will only give permission for using their browser features to trustworthy websites. However, if the Feature Policy is not properly configured by the website developer, these permission requests may be coming from third-party embedded iframes. For example, advertisements are an important constituent of the Web. Many websites publish ads to generate revenue. One possible way to publish an ad is to embed iframes into the main website. The content of this iframe will be provided by the advertisement network or the landing page for the advertisement [16].

Figure 1 provides an example attack scenario for the type of vulnerability studied in this paper. First, Alice sets her site’s Feature Policy as a wildcard for the camera browser feature (Step 1). Eve serves malicious advertisements (Step 2) and Alice publishes Eve’s ads on alice.com (Step 3). After that, Bob visits alice.com and clicks on “allow” to access the feature when the browser prompts a dialog box for permission (Step 4). Since alice.com set the Feature

![image](image-url)

**Fig. 1:** A Simple Attack Scenario: 1. Alice serves a website in which Feature-Policy is set as a wildcard for camera feature. 2. Eve serves malicious advertisements. 3. Alice publishes Eve’s ads on alice.com by embedding an iframe. 4. Bob visits alice.com. Browser prompts a permission dialog that contains misleading information which says feature request is coming from alice.com whereas it originates from eve.com. Bob clicks “allow” on the dialog box, 5. Eve uses the feature and steals sensitive data.

![image](image-url)

**TABLE I:** Possible Feature Policy Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ambient-light-sensor”</td>
<td>“autoplay”</td>
</tr>
<tr>
<td>“camera”</td>
<td>“accelerometer”</td>
</tr>
<tr>
<td>“encrypted-media”</td>
<td>“fullscreen”</td>
</tr>
<tr>
<td>“geolocation”</td>
<td>“geolocation”</td>
</tr>
<tr>
<td>“gyroscope”</td>
<td>“magnetometer”</td>
</tr>
<tr>
<td>“microphone”</td>
<td>“microphone”</td>
</tr>
<tr>
<td>“midi”</td>
<td>“picture-in-picture”</td>
</tr>
<tr>
<td>“speaker”</td>
<td>“sync-xhr”</td>
</tr>
<tr>
<td>“usb”</td>
<td>“vr / xr”</td>
</tr>
</tbody>
</table>

**Table:** Possible Feature Policy Directives
Policy for camera as a wildcard, eve.com also has the camera permission and Eve has access to Bob’s camera (Step 5). Note that for our attack scenario to succeed three conditions need to be met: the Feature Policy has to be set to a wildcard, the malicious iframe has to be embedded in the main website and user has to click on “allow” on the dialog when prompted. To measure the possibility of our attack scenario in the wild, we crawled the Majestic top 1 million list and found the websites that use wildcard in their Feature Policy. We explain our measurements in the wild in detail in Section V.

III. Methodology

We study the privacy issues resulted from Feature Policy misuse by following two phases:

1) Design the test cases that will cover possible iframe inclusions to the main website.
2) Visit the main website with different Web browsers to examine browser behavior.

In the remainder of this section, we describe these two phases in detail.

A. Design the Test Cases

In this paper, we aim to identify dangerous browser behaviors made possible by iframe inclusions to the main website and by two different settings of the Feature Policy, namely, not setting the Feature Policy at all and setting it to a wildcard. To this end, we designed three test cases to be tested on each Web browser. To perform our test cases, we created three websites with different origins. The reason to use different origins for all three websites is to demonstrate iframes as embedded third-party contexts. From here on we will use the terms mainsite, iframesitel, and iframesite2 for the main website, the first iframe source, and the second iframe source, respectively. Mainsite has two embedded iframes in its HTML source code with the source set as (iframesitel) and (iframesite2). Iframesitel and iframesite2 have the exact same HTML source code which tries to access and use the geolocation, camera or microphone feature, depending on which feature we are testing. However, the two iframes are hosted under different origins to represent different third-parties. The three main test cases are summarized as follows:

1) Test1: The feature is only used by iframesitel and not by mainsite.
2) Test2: The feature is used by mainsite, a permanent permission is granted by the browser, then the same feature is used in an iframe of mainsite with the source as iframesitel.
3) Test3: The feature is used in an iframe of mainsite with the source set as iframesitel, a permanent permission is granted by the browser after mainsite is visited and “allow” is clicked in the dialog box, then the feature is used in another iframe of mainsite with the source set as iframesite2.

Our main concern when designing the test cases was to cover different types of iframe inclusions to the main website. Specifically, in our first test case (Test1) we include an iframe (iframesitel) to mainsite. In Test1, the feature request originates from the iframe (iframesitel) and not from mainsite. In the second test case (Test2), we include the iframe (iframesitel) after the user approves the feature request originating from mainsite itself. In the third test case (Test3), we include a second iframe (iframesite2) after the user approves the feature request originating from the first iframe (iframesitel). This way we aim to observe whether permissions given to different origins contained in the same website (mainsite) affect each other. To be more precise, as the user, we grant permission to the feature requesting origin when browser prompts (whether this request is originated from the mainsite itself or any other included iframe). After granting the permission, we try to see if the browser forwards this permission to another iframe source’s origin that resides in the mainsite without the user explicitly approving a request for that iframe source’s origin.

We also want to observe the fidelity of the information given in the dialog box prompted by the browser. In Test1, since the feature is only used by the iframe (iframesitel) and not by the mainsite, we expect to see the origin of the iframe (iframesitel) inside the permission dialog. In Test2, we expect to see the origin of the mainsite in the first dialog and the origin of the iframe (iframesitel) in the second dialog, after the browser grants permanent permission to the main website. Similarly in Test3, we expect to see the origin of the first iframe (iframesitel) and later on the origin of the second iframe (iframesite2) in the prompted dialog. In each test case we also expect to see the name of the correct feature inside the dialog box. We chose the three test cases to cover a group of such scenarios mentioned above.

We apply the three test cases for three browser features which are geolocation, camera and microphone. Furthermore, we applied the three test cases both without setting the Feature-Policy header in the main website and with setting the header to wildcard for all three features. In total, we tested each browser for 18 different conditions.

B. Visit Mainsite with Different Web Browsers

When a user visits a website from a Web browser, if any feature is used in the home page of that website, the browser prompts a permission dialog which asks the user to allow/deny the website’s feature access requests. Since in our websites we are using the feature on the home page, we were able to see the prompt immediately after visiting the mainsite.

For each of the 18 conditions, we created a mainsite under the same domain. These mainsite instances have small differences such as the Feature Policy header setting, the included iframes, the used features, to comply with the test cases and conditions defined in Section III-A. We show the differences between these mainsite instances in Table II. As it can be seen, there are ten different mainsite instances, however we created three groups of websites with same mainsite configuration (from mainsite1-6a) for each of the three features. Therefore, in total we created thirty mainsite instances.
For Test1, we visited the mainsite1 and waited for the dialog to be prompted and if prompted we clicked “allow” to observe whether the feature can actually be used by the iframe. For our second test case we first visited mainsite2 and then when prompted we accepted the request on the browser. For the browsers that offer an option to record this choice and persist the permission we also selected that option. In order to see whether the iframe can use the feature without the user’s explicit approval, even though mainsite does not use it anymore we visited mainsite2a where feature request originates from iframesite1 and not from mainsite. For the last test case, we visited mainsite3 and then when prompted we accepted the request on the browser. For the browsers that offer an option to record this choice and persist the permission we also selected that option. After that we visited mainsite3a which excludes iframesite1 and includes another iframe which is iframesite2 that tries to access the feature inside the mainsite. To test the wildcard configuration of Feature Policy, we followed the same steps with mainsite4-6a.

While visiting mainsite for all test conditions, we investigated the browser’s behavior. Specifically, we recorded the text displayed inside the permission dialog and examined to see if the information in the text is enough to give the user an understanding about the permission to be granted (contains the correct feature and correct origin where the feature request originates). Browsers hold a list of allowed websites for every feature. We also recorded the change in those lists after we select options inside the prompts to give permanent permissions.

### IV. Evaluation

We evaluated the approach discussed above on the following 9 browsers: Chrome 77, Firefox 69, Chromium 71, Opera 64, Brave 0.55, Edge 42, Apple Safari 10.12, iOS Safari 10.11, Android Chrome 77. A browser is considered vulnerable if for any of our test cases it behaves in such a way that contributes a malicious third-party website to steal sensitive data. After analyzing our test results, we found that the following five Chromium based browsers are open to social engineering attacks based on our attack model: Chrome, Chromium, Opera, Brave, and Android Chrome. According to StatCounter [17], the market share of Chrome desktop browser is 70.71% and for the Chrome mobile browser is 60.52%, which means that the majority of Web users utilize partially-vulnerable browsers. For other browsers we grouped the ones that behave the same way together. In Table III, we present only the vulnerable part of our results. We omit the results for the other browsers for space reasons, since they did not present any vulnerability related to the Feature Policy. In the following, we classify the browsers according to their reactions to our test cases and we describe the behavior that we encountered for each class of browsers in detail.

#### A. Chromium Based Browsers

Chromium, Chrome, Opera, Brave, and Android Chrome are found to be partially-vulnerable and acted the same way against our tests since they are all Chromium based browsers. During our experiments we observed that Chromium based browsers always show the permission dialog displaying the domain of the mainsite independent of the actual origin trying to access the feature. When the Feature Policy is set as wildcard for a feature, any iframe that is embedded in the mainsite is able to use the feature even if the permission is only granted to the mainsite. This is the expected behavior. If the request is made from an iframe and not the mainsite, then, prompting for the mainsite may be misleading the user. For example, in Table III, it can be seen that for Test1 where the feature is only used by the iframesite1 and not by the mainsite Chromium based browsers prompted a dialog with the name of the mainsite. When the user clicks “allow” in this dialog, Chrome recorded the origin of mainsite under permitted websites for that feature. In this case mainsite can be a trustworthy website and iframesite1 can be a malicious advertisement. The user who trusts the mainsite is likely to click “allow” when prompted with the misleading dialog. However, Chromium developers consciously chose to grant permissions to the top-level in response to several consumer surveys and user studies which are not publicly available. The developers explained that the reason behind this trade-off is to prevent confusing users, because most of them are likely unaware of the composed nature of websites [18]. However, we believe that this design choice of Chromium-based browsers limits the information given to user and thus, does not provide user with a full picture to “allow/deny” the request.

#### B. Edge

Edge does not support the Feature Policy. However, it is not vulnerable against our threat model because anytime an iframe tries to access a browser feature, Edge prompts a dialog that asks permission for the correct origin (i.e. the actual origin that is requesting feature permission). On the other hand, when a website developer uses the policy to limit feature access from iframes, their measure will be rendered useless for this browser. Website developers would think that they restricted feature access from iframes where in fact, feature access permission will depend on the user’s decision after the browser prompts a dialog box. If we relate this case to the previous example in Section IV-A, Edge will ask for permission for iframesite1, which is the malicious advertisement domain and the user will have a choice to click “allow” or “block” in the permission dialog seeing the origin of the malicious website in the dialog.
null
concept code as an advertiser on other ad networks (popcash.net, infolinks.com etc.) and observed that these ad networks also do not allow custom code from advertisers.

4) As previously mentioned in Section II-C, our attack scenario could be successful in the wild if an attacker is in control of an iframe source embedded in any extracted website (extracted websites are mentioned in 1). To this end, an attacker could register a non-existing or expired domain (i.e., a NXDOMAIN) that is currently used by extracted websites as an iframe source, serve their malicious code, and wait for users to click “allow” when prompted. To find out if this attack could be performed on the extracted websites, we visited each of the extracted websites and collected the DNS traffic with Wireshark [26], then filtered the output to only have NXDOMAINs. After that, we cross-referenced the NXDOMAINs with the output of the modified extension to find whether the requests to NXDOMAINs originated from iframes. Finally, we observed that most of the NXDOMAIN requests are made from scripts that are in HTML source code of extracted websites and not in iframe sources. Consequently, we could not demonstrate our attack scenario on real-world websites.

VI. COUNTERMEASURES

To prevent the unwanted usage of browser features that may cause sensitive data leaks, both website developers and browser developers can take measures. From the website developer’s perspective the most basic approach would be setting the Feature Policy to only allow same-origin accesses which will prevent third-parties to access the features. Another approach could be setting the allow attribute for iframes. However, the Feature Policy header makes configuration easier in the sense that allow attribute has to be applied to each iframe individually whereas setting the Feature Policy header automatically defines the rules for the whole website. Thus, it prevents human error such as missing some iframes in the webpage. It should be noted that even just missing one single iframe here would be enough for our attack model to succeed. Moreover, by using the Feature Policy header a developer can only allow the features that the website needs to function and set others as “none” to prevent possible leaks. For example, if a website only needs location data to function, the Feature Policy header for geolocation as “self” which is equal to allowing only same-origin and setting others as “none” would be the most secure approach assuming this policy is handled correctly by the browser.

From a browser developer’s perspective, on the other hand, not persisting permissions for third-party websites could be a good solution against the possible problem caused by attackers re-registering NXDOMAINS which are already used as sources of iframes in several websites. Also in this paper we discovered that some of the browsers display prompts with misleading information. The prompts displayed by browsers should be accurate to make users understand what they are permitting when they click “allow”.

VII. DISCUSSION

In this section, we first discuss the limitations of our study. We then talk about possible directions for future work.

A. Limitations

We did not test older versions of browsers that may still be used by some users and they possibly do not have support for the Feature Policy since it is fairly new and a work in progress. However, most modern browsers have an auto-update setup by default so the majority of users likely run a recent version of a given browser. For example, as of September 2019 9.89% of users use Chrome 77, 25.26% use Chrome 76 whereas only 2.38% use Chrome 75 [27].

We only crawled the homepages of the top 100K Majestic websites. Therefore, we did not investigate whether there are iframes on subpages that are potentially vulnerable or not.

We did not investigate some of the advertisement networks for several reasons such as some of them required payment to register or we were able to submit a registration request but it is not verified.

B. Future Work

In this paper, we showed that the Feature Policy is not yet fully understood or adopted by Website and browser developers, and that this opens the possibility for misconfigurations and bugs that lead to privacy leaks. While we performed a measurement of the top 100k websites according to majestic, however, we could not find evidence of malicious parties taking advantage of this attack in the wild. As part of future work, we plan to perform more extensive measurements, looking not only at the home pages of websites but also at subpages, to better understand whether sensitive browser features are being accessed by untrusted parties.

VIII. RELATED WORK

Related work studied vulnerabilities in Web browsers that can be exploited by using iframes [28], [29]. Another research found that because of the browser failing to ask permission, websites were able to read the battery status and profile users accordingly [30]. Previous studies [8], [9], [10] focused on adoption of response headers in general or in particular such as Content Security Policy [11], [12] and Referer Policy [13]. Studies showed that unless these policies are used correctly both by server-side and client-side, they do not protect against possible attacks. Other research investigated script inclusion to steal sensitive data [14], [15] and proposed sandboxing scripts as a countermeasure. [31] showed that third-party scripts included in websites may cause sensitive data leakage.

IX. CONCLUSION

In this paper, we analyzed 9 Web browsers and their behavior against different configurations of Feature Policy and iframe inclusions. We created 3 websites and in total 18 test cases for each browser. The attack scenario we present in this paper is possible for every browser. However, we found that Chromium based browsers presents misleading information
which will in turn make the user more open to our attack scenario. We also showed that adoption of Feature Policy has not been completed by most of the browsers since we observed inconsistent behavior (e.g., not supporting `geolocation` feature while supporting `camera` and `microphone` features) and no support at all (Edge).

REFERENCES


A. Feature Policy Option Usage

Fig. 2: Feature Policy Option Usage for Features in Majestic top 100K.