# On the Perils of Leaking Referrers in Online Collaboration Services

Beliz Kaleli, Manuel Egele, and Gianluca Stringhini

Boston University
{bkaleli,megele,gian}@bu.edu

**Abstract.** Online collaboration services (OCS) are appealing since they provide ease of access to resources and the ability to collaborate on shared files. Documents on these services are frequently shared via secret links, which allows easy collaboration between different users. The security of this secret link approach relies on the fact that only those who know the location of the secret resource (i.e., its URL) can access it. In this paper, we show that the secret location of OCS files can be leaked by the improper handling of links embedded in these files. Specifically, if a user clicks on a link embedded into a file hosted on an OCS, the HTTP Referer contained in the resulting HTTP request might leak the secret URL. We present a study of 21 online collaboration services and show that seven of them are vulnerable to this kind of secret information disclosure caused by the improper handling of embedded links and HTTP Referers. We identify two root causes of these issues, both having to do with an incorrect application of the Referrer Policy, a countermeasure designed to restrict how HTTP Referers are shared with third parties. In the first case, six services leak their referrers because they do not implement a strict enough and up-to-date policy. In the second case, one service correctly implements an appropriate Referrer Policy, but some web browsers do not obey it, causing links clicked through them to leak their HTTP Referers. To fix this problem, we discuss how services can apply the Referrer Policy correctly to avoid these incidents, as well as other server and client side countermeasures.

**Keywords:** Web Security · Online Collaboration Services · Referrer Leaking · File Sharing.

## 1 Introduction

Collaboratively editing and sharing documents, presentations, and spreadsheets online have become an increasingly popular aspect of the modern-day workflow. To facilitate this, so-called online collaboration services (OCS) emerged and allow users to simultaneously create, edit, and share documents through their web browsers. A characteristic feature of many of these services is the capability to share access to a specific document via a *secret URL*. The security and secrecy of the content maintained in the shared resource hinges on the

*secret URL* to only be known by authorized collaborators. That is, should the *secret URL* be disclosed to unauthorized entities, the secrecy of the collaborative content is compromised.

In this paper, we show that the secret location of an OCS document is frequently leaked through HTTP Referers, even in situations where the OCS platform explicitly tries to restrict such leakage. The problem arises from how modern web browsers handle HTML hyperlinks (or links). By default, any link the user clicks on will trigger an HTTP request to the target of the link. Furthermore, this HTTP request includes an HTTP header field (the `HTTP_REFERER`), that indicates the URL of the document that contained the link that was clicked. By default, if the shared content is hosted under the *secret URL* and an authorized user clicks a link embedded in the shared document, the browser will leak the *secret URL* as the value of the `HTTP_REFERER` header to the target of the link (i.e., an unauthorized entity). With knowledge of the *secret URL*, the unauthorized entity can now access, and potentially modify, the content hosted on the OCS platform. Since OCS platforms are widely-used by both individuals and companies, uploaded documents may contain a variety of sensitive information, making this threat even more serious. As referrers can include sensitive information, three readily available countermeasures exist that allow web site operators to control how and whether browsers send HTTP Referer headers. First, as specified by RFC2616[19] in 1999, browsers have always stripped referrer headers when transitioning from content served via HTTPS to content served via HTTP. Second, if a link includes the `noreferrer` relation (i.e., `rel="noreferrer"`), the browser will also omit the referrer header in any request triggered by following the respective link. Third, and most recently, the W3C published the Referrer Policy [8] Candidate Recommendation in 2017. While the first two options categorically remove referrer request headers, the Referrer Policy allows web site operators to control, in a more fine-grained manner, whether and how browsers should include information on the referrer in HTTP requests. Correctly implemented and deployed, the Referrer Policy can prevent the leakage of sensitive information through HTTP Referer headers. Unfortunately, as this paper shows, the incomplete implementation of the policy in browsers and the sub-optimal choice of policy server-side both can lead to catastrophic disclosure of the *secret URL* on popular OCS.

Previous studies focused on services that allow users to share resources [18] and downloading content [25] via URLs. These papers show that *secret URL*s are often used to share resources, for example in the case of illegal movie streaming [18] or other illegal content [25]. Other papers discussed usage patterns of file-sharing services [13], the security and privacy of services providing *secret URL*s [15], and HTTP referrer leaks in referrer anonymizing services [30]. Antoniades et al. [13] presented a detailed study of one-click hosters traffic and services but did not focus on the security perspective. Hutchinson et al. [15] studied the security of *secret URL*s generated on file hosting services, but they did not investigate the possibility of leaking these URLs. Referrer leaks are studied by Nikiforakis et

al. [30] without associating them with online collaboration services. Moreover, Nikiforakis et al. [30] predate the release of the W3C Referrer Policy Candidate Recommendation by five years and hence the impact of the policy or its incorrect implementation in browsers was not studied in that work. To the best of our knowledge, we are first to study the security implications of referrer leaks to the users of online collaboration services.

To study the extent of the problem, we present a methodology to identify whether a given OCS is vulnerable to the HTTP Referer leak sketched above. We perform an extensive study on 21 different online collaborative services and find that seven of them are vulnerable. As a correct enforcement of Referrer Policy requires the correct choice of policy (server-side) and its correct implementation (client-side), we evaluate all services with 6 different web browsers. Through this assessment, we identify two root causes that lead to the leakage of *secret URL*s from popular OCSs. Both causes are related to the incorrect application of the Referrer Policy. In the first case, we find that six services do not set a Referrer Policy at all. This results in web browsers implementing the overly permissive policy that only strips referrers from requests that transition from HTTPS to HTTP. In the second case, we find that one service (Overleaf) sets the Referrer Policy correctly, but the incomplete support of the Referrer Policy in two web browsers we tested(Edge and iOS Safari) results in the leakage of the *secret URL*. While a comprehensive solution to prevent the leaking of sensitive information through HTTP Referers, is most realistic by having browsers implement the W3C Referrer Policy, this paper also discusses additional effective countermeasures that can be deployed in the meantime.

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

- We present a security vulnerability arising from the improper handling of HTTP Referer information in files shared on OCSs. The vulnerability manifests in a third party learning the *secret URL* which provides access to a perceived securely-shared document.

- We systematically identify and evaluate 21 online collaboration services with six different web browsers and identify OCS and browsers that are vulnerable to this kind of information disclosure.

- Our analysis shows that seven of the analyzed Online Collaboration Services are vulnerable to the identified information leak. Six services are vulnerable due to the overly permissive default policy, and one is vulnerable because browsers incorrectly implement the Referrer Policy as specified by the W3C.

- We discuss how existing server and client side mitigations can be used to correctly deal with the HTTP Referer leak problem.

## 2 Background

In this section, we provide the pertinent background information that the remainder of the paper relies upon.

### 2.1 Online Services Allowing File Sharing

A number of services allow users to share files, for example, file hosting providers and instant messaging services. Most services also provide tools to edit these files online. In some cases, given the proper permissions, multiple people may work on the same document at the same time. These permissions can be given through sharing options (e.g., read only, write).

Some of these services allow users to share access to the file by means of a unique and perceived *secret URL*. By selecting the appropriate option, the service generates a unique link that can then be shared with other users. URLs can have permission associated with them (i.e., write or read-only), and the knowledge of the URL grants access to the resource. For example, when a permission is set to editable, anyone who has that URL can edit the document but if the permission is set to read-only, the link only provides read access to that file. The editable and read-only links have different unique identifiers. Some services also let the user set an expiration date or time limit to those links which are an extra layer of security. These *secret URL*s generally have the following form:

https://www.service-name.com/files/<`UniqueIdentifier`>

In the above example, the "UniqueIdentifier" part refers to the specific file, thus, it is different for each file that is shared. The generation of that unique identifier can be random or sequential for each time a user shares a file. The identifier may contain the file name and the length of that identifier varies by service [29]. Ideally, this identifier should not be guessable to prevent an adversary to access the resource.

### 2.2 HTTP Referer

HTTP requests and responses begin with HTTP headers, and requests contain, among other headers, a `HTTP Referer` field. The HTTP Referer identifies the URI from which the request originated. When a user clicks on a link in her web browser, that request is sent to the destination webpage. The HTTP Referer field in the request holds the URI from which the user clicked on the hyperlink. If HTTP Referer fields are logged by the visited web server, the administrators can identify the webpage from which the user is visiting their website. The information that resides in HTTP Referer can be used to personalize the website such as providing specific help, suggesting relevant pages to targeted users [24], generating special offers, webpage analytics (ex: analyzing where most of the

traffic is coming from) [35] or to block visitors from specific domains. As pervasive use of the HTTP Referer raises privacy and security concerns, the way browsers treat the HTTP Referer is configurable.

### 2.3 Existing Mitigations

| HTTP Referer | Referrer Structure |
|---|---|
| No Referrer | |
| ASCII Serialized | https://www.service-name.com/ |
| Full Referrer | https://www.service-name.com/files/UniqueIdentifier |

**Table 1.** HTTP Referer structures for files in online collaboration services.

Two main mitigations were proposed to prevent referrer leaks: the Referrer Policy and specifying "noreferrer" as an HTML link type. In the following, we describe these two approaches in detail.

*Referrer Policy.* The Referrer Policy governs which information should be included in the HTTP Referer header when fetching sub-resources, prefetching, or performing navigation [8]. The W3C currently specifies nine specific of referrer policies that affect browser behavior with respect to HTTP Referer headers. Those behaviors mostly differ between same-origin and cross-origin requests. If a *request*'s origin is not the same as the *request's current URL*'s origin, the request is called cross-origin. Referrer policies are defined in such a way that the potential trustworthiness of the source and destination websites are considered. For example, the `"strict-origin"`, `"no-referrer-when-downgrade"`, `"no-referrer"` and `"strict-origin-when-cross-origin"` policies specifically state that if a request is sent from a TLS-protected environment to a *non-potentially trustworthy URL* the referrer header will be completely omitted. The assessment technique for a URL's trustworthiness can also be found on the W3C website [8]. The Referrer Policies currently defined by W3C can simply be explained as follows.

- `"no-referrer"`: Referrer header is omitted entirely for requests to any origin.
- `"no-referrer-when-downgrade"`: Full referrer is sent in requests from a TLS protected environment to a *potentially trustworthy URL* and also from a non-TLS protected environment to any origin. Conversely, referrer header is omitted in requests from a TLS protected environment to a *non-potentially trustworthy URL*.
- `"same-origin"`: A full URL, stripped for use as a referrer (the algorithm to strip URLs is defined in [8]), is sent within requests to same-origin. However, the referrer header is omitted in cross-origin requests.

- `"origin"`: Along with both same-origin and cross-origin requests, an ASCII serialization of the referrer is sent. An example of this serialization result is given in Table 1.

- `"strict-origin"`: ASCII serialization of the referrer is sent along with requests from a TLS protected environment to a *potentially trustworthy URL* and from a non-TLS protected environment to any origin. Whereas, no referrer is sent from a TLS-protected environment to a *non-potentially trustworthy URL*.

- `"origin-when-cross-origin"`: A full URL, stripped for use as a referrer, is sent within requests to same-origin. ASCII serialization of the origin of the request is sent within requests to cross-origin.

- `"strict-origin-when-cross-origin"`: A full URL, stripped for use as a referrer, is sent within requests to same-origin. For the cross-origin requests, the same schema is applied as `"strict-origin"`.

- `"unsafe-url"`: A full URL, stripped for use as a referrer, is sent within both same-origin and cross-origin requests.

Without any declared referrer-policy attribute, a HTML `<a>` element's referrer policy is the empty string. Thus, navigation requests initiated by clicking on that `<a>` element will be sent with the referrer policy of the `<a>` element's node document. If that document has the empty string as its referrer policy, the algorithm will treat the empty string the same as `"no-referrer-when-downgrade"`.

The Referrer Policy is a useful mechanism that a web service has to signal to the web browser how it should deal with referrers on its service. To be effective, however, the web browser needs to implement the Referrer Policy correctly, and in particular deal with all the possible policies listed above. In this paper, we show that this is not always the case, resulting in some OCSs leaking referrers on certain browsers, despite using the Referrer Policy correctly.

*HTML Link Type.* The HTML5 standard added support for the `noreferrer` relation attribute for links (i.e., `rel="noreferrer"`). Referrer headers will be suppressed for links that have the `noreferrer` relation attribute set [8]. Hence, no referrer information will be leaked when following such a link [9].

The behavior of the referrer header in an HTTP request is adjustable on both the client and server side. If these optional HTTP Referer fields are not adjusted properly by the online service or the user, this will cause a potential leak of the location (i.e., URL) of the secret resource. Since access to the resource is only determined by the knowledge of such a URL, this can have security repercussions. This vulnerability is the main concern of this paper.

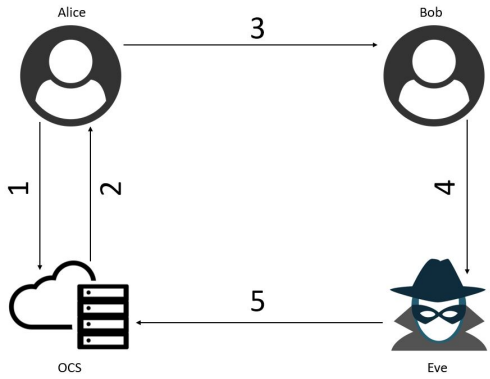


**Fig. 1.** A Simple Attack Scenario: 1.Alice uploads a file, file has a link to Eve's website `evil.com`, 2. Generates the *secret URL*, 3. Alice shares file with Bob, 4. Bob opens resource, sends HTTP Request to Eve's website `evil.com`, 5. Eve gets access to the resource through referrer

### 2.4 Threat Model

Online Collaboration Services are popular among individuals and companies alike. Therefore, a significant number of files are uploaded and shared through those services. Because of the ease of sharing documents that these services provide, companies use them to collaborate on documents. Therefore, these documents are likely to contain confidential information. For example, personal information of customers can be kept in a document where authorized employees constantly add new customer information such as social security numbers, phone numbers, email addresses, etc. If such a document got leaked, miscreants could use this information to perform identity theft or other malicious activity [27]. In the case of individual users, people can store any kind of sensitive data in OCS such as bank account details, passwords, photos, payment details, real estate information, etc.

Knowing the *secret URL* of a resource allows anyone to access it (and possibly edit it). The reason for generating such URLs is to easily share a resource. Only owners and users that the owners shared the resource with should know the *secret URL*. These URLs are mostly generated uniquely and in a non-guessable fashion because of security concerns. However, these secret links grant direct access to a resource therefore if they got leaked, simply visiting that URL would provide the attacker access to the secret resource. According to the permissions of that link (editable or read-only), an attacker can steal the confidential information on that secret resource or modify/delete it.

Figure 1 provides an example attack scenario for the type of vulnerability studied in this paper. First, Alice uploads a file to an Online Collaboration Service, and the file contains a link to `evil.com` (Step 1). The online service then generates

a *secret URL* (Step 2) and Alice shares the *secret URL* with Bob (Step 3). After that, Bob opens the resource and clicks the link in it (Step 4). By clicking the link in the file, an HTTP Request is made to `evil.com`. That request contains HTTP Referer field which reveals the unique URL to the specific file. This URL is not supposed to be known by people other than the owner and ones that the owner shared at the first place which is Alice and Bob. Assuming that no client or server side mechanism had omitted or trimmed the HTTP Referer part of the request and Eve, the owner of `evil.com` logs the HTTP Referers of requests, which is often the case on the Web [12], then the specific file becomes vulnerable to attacks because the unique URL of the file is recorded in `evil.com`'s server (Step 5). Eve becomes capable of compromising this information to read and possibly edit the specific file, depending on the permissions set by Alice.

## 3 Methodology

As discussed, in this paper we aim to quantify how prominent the referrer leak problem is in online collaboration services. To this end, we analyzed 21 different online collaboration services. Our study consists of the following four phases:

1. Defining relevant services
2. Creating files
3. Sharing files
4. Examining the referrer

In the rest of this section, we describe these four phases in detail.

### 3.1 Defining Relevant Services

Our main concerns while selecting a set of services to study were threefold: (i) each online service should allow creating or uploading a file, (ii) links embedded in that file should be clickable, and (iii) access to the file can be delegated by sharing a URL. First, we investigated the most popular online services through Google queries and Alexa top lists [1]. For Google queries we searched for keywords such as: online file sharing, online collaboration tools, file storage and sharing services etc. We implemented a crawler to crawl Alexa categories on February 13, 2019. We searched categories that contain popular online collaboration services found from Google queries (such as docs.google.com, dropbox.com etc.). Then, we manually tested all the services that belong to these categories to see if they were suitable for our experiment. We found that most Online Collaboration Services offer the three functionalities outlined above. We found a total of 67 online services as part of our investigation. In order to understand whether those services comply with our criteria, first we set up an account on

each of them. Then, with this account, we tried uploading a supposedly secret document with an embedded URL that points to a server under our control. After uploading, we checked if the embedded link was clickable. Finally, we tested the share option for the resource to see if the service allows users to share files via a unique URL. The above procedure eliminated 46 services [1] because of the following reasons: only video meetings can be shared with URL, share option is unavailable in free version, link in a document is not clickable, no free version, sign up needs credit card information, documents are only shared via email, files can only be downloaded, open option only downloads the file, site cannot be reached, or account confirmation email is not sent by website.

After eliminating these 46 services, we were left with 21 services, consisting of a mix of file hosting services and instant messaging services. A full list of the services used in this study is available in Table 2.

### 3.2 Creating Files

As mentioned before, some providers only allow a file to be shared among users who have accounts on that service. Therefore, for examining online collaboration services, we created two different accounts on each service. One account was responsible for uploading a file and getting the unique link that belongs to that file, whereas the other account was used to open the shared file online by using the unique URL and clicking the link in the file.

Services allow different types of files to be uploaded or created. As part of our study, we created multiple types of files on each of the 21 services. These file types were selected among those that are frequently used in a collaborative environment, for example ".doc", ".docx", ".pdf", ".xls", ".xlsx", ".ppt", ".pptx", or ".note" files. For file hosting providers and instant messaging services that offer the feature, files were uploaded under the first testing account.

We embedded a link referring to our servers in all the files we uploaded to the services. For our experimental purposes, uploaded files contained nothing but the URL to our webpage. In a real-world scenario, and in general, this link would be a third-party website and the file would have some other data than the

[1] Ruled out services are: `prezi.com`, `uploaded.net`, `4shared.com`, `1fichier.com`, `filerio.io`, `filefactory.com`, `bibsonomy.org`, `adrive.com`, `drivehq.com`, `clickability.com`, `filesanywhere.com`, `livedrive.com`, `smartfile.com`, `elephantdrive.com`, `mydocsonline.com`, `jungledisk.com`, `kontainer.com`, `mozy.com`, `exavault.com`, `thinkfree.com`, `cryptoheaven.com`, `powerfolder.com`, `filesave.me`, `crocko.com`, `cloudsafe.com`, `trueshare.com`, `diino.com`, `filehostname.com`, `file-works.com`, `wonderfile.net`, `classlink.com`, `signiant.com`, `fileflow.com`, `bluejeans.com`, `dropsend.com`, `hightail.com`, `justcloud.com`, `sugarsync.com`, `idrive.com`, `sharepoint.com`, `transfernow.com`, `deliveryslip.com`, `mango.com`, `ionos.com`, `mediafire.com`, `tresorit.com`, `sync.com`.

link. The embedded URL refers to a PHP script which simply logs all HTTP Header information (including `HTTP_REFERER`) of each incoming HTTP request. This way we were able to easily understand if the HTTP Referer is leaked by each OCS.

### 3.3 Sharing the Files

Following the threat model presented in Section 2.4, we only considered OCSs that allowed users to share content via a unique *secret URL*. A unique URL is created by providers when we click on the share via link option. Some services provide different URLs to the same file that have varied privileges such as non-editable and editable. In order to span all possibilities, we created both types of links to each file. Then we shared these links with the second account that we set up. Note that not every online service requires a valid account to view the file through a shared unique link. However, some of the instant messaging services did not allow uploading files. For this situation, we sent the created files directly to a chat between our first and second testing account.

### 3.4 Examining the HTTP Referer

To assess whether a given OCS leaks the *secret URL* via referrer headers, we visited the shared document and clicked on the link. As the browser implementation might affect how referrers are treated, we visited each shared document (and included link) with six different browsers. For this experiment we specifically used the latest stable versions (at the time of writing) of the following browsers: Google Chrome (version 72), Mozilla Firefox (version 65), Microsoft Edge (version 42), Safari (version 12), Mobile Chrome (version 71), and iOS Safari (version 12). By following the unique links, the original shared file is opened in each browser. Some providers required the user that tries to open the unique link to have an account on that service. In this case, we used the second account we set up to display the original file. On some instant messaging services the file is opened just by clicking on it in the chat. After loading the file, we clicked on the embedded link that refers to our PHP script. In our experiment, the PHP script that was the destination of the link displayed the HTTP Header information. In a real scenario, after this click, the user would get redirected to the third party webpage that is the target of the link.

As part of the HTTP headers, we also recorded the HTTP Referer. HTTP Referers can be divided into three categories: completely omitted (`"no-referrer"`), ASCII Serialized to show only the website name, and contains full URL information. The possible referrer structures are shown in Table 1.

Different web browsers gave different output for the same unique URLs. This is due to the fact that browsers have different settings for handling HTTP Referers.

The secret links generated by online services are opened with six different web browsers. Then, the link to our servers inside the documents are clicked and output of our PHP script is observed. Available Referrer-Policies handle referrers as threefold as shown in Table 1. The original resource can only be accessed if the leaked referrer is a full referrer and the same as the *secret URL*. Therefore, the outputs that show no HTTP Referer and only show an ASCII serialized referrer which contains only the host part of the unique link are recorded as not vulnerable. The remaining HTTP Referers which contain full URLs are cause for suspicion and opened with each web browser. The cases where the resulting page shows the original document are recorded as vulnerable. If instead the secret file cannot be accessed by using the referrer link the service is recorded as not vulnerable.

## 4 Evaluation

We evaluated the approach discussed above on 21 different online collaboration services. This evaluation identified that seven out of 21 services are vulnerable and leaked the HTTP Referer to unauthorized third parties. A service is considered vulnerable if visiting the leaked referrer results in opening the original resource. After analyzing the leaked HTTP Referers for all services, we found the following seven services for which our attack succeeded: Google Docs, Onehub, Overleaf, Box, Flock, Evernote, and Linkedin Slideshare. For six services we identify the source of the vulnerability in the service not implementing the Referrer Policy correctly, while for one service the issue lies in the fact that some web browsers do not react correctly to the Referrer Policy declared by the website. Table 2 reports a summary of the vulnerabilities that we encountered in our study. In the following, we discuss the issues discovered for each of the services in detail.

### 4.1 Google Docs

Our initial investigations found that Google Docs leaked referrers to third-party entities. Google acknowledged our findings, promptly fixed the underlying problem (via `rel="noreferrer"`). However, their prompt action prevented us from testing all browsers against their services. Thus, in table 2, browsers that could not be tested with Google Docs are assigned yellow squares which imply that the test was not applicable.

### 4.2 Overleaf

Overleaf is a collaborative writing and publishing service which is widely used for producing academic papers. Overleaf offers templates for different types of

| Service Name | Google Chrome | Microsoft Edge | Mozilla Firefox | iOS Safari | Apple Safari | Android Chrome |
|---|---|---|---|---|---|---|
| Google Docs | ⚡ [5] | □ | ⚡ [5] | □ | □ | □ |
| Office365 | ○ | ○ | ○ | ○ | ○ | ○ |
| Zoho Cliq | ○ | ○ | ○ | ○ | ○ | ○ |
| Dropbox | ○ | ○ | ○ | ○ | ○ | ○ |
| ShareFile | ○ | ○ | – | ○ | ○ | – |
| SeaFile | ○ | ○ | ○ | ○ | ○ | ○ |
| Box | ⚡ [1] | ⚡ [1] | ⚡ [1] | ⚡ [1,2,3] | ⚡ [1] | ⚡ [1,2,3] |
| Onehub | ⚡ [4] | ⚡ [4] | ⚡ [4] | ⚡ [4] | ⚡ [4] | ⚡ [4] |
| pCloud | ○ | ○ | ○ | ○ | ○ | ○ |
| Overleaf | ○ | ⚡ [2,5] | ○ | ⚡ [2] | ○ | ○ |
| OpenDrive | ○ | ○ | ○ | ○ | ○ | ○ |
| Fleep | ○ | – | ○ | ○ | ○ | □ |
| Slack | ○ | ○ | ○ | ○ | ○ | ○ |
| Hangouts | – | – | – | ○ | □ | □ |
| Flock | ○ | ○ | ○ | ⚡ [2] | ○ | □ |
| Mega.nz | ○ | ○ | ○ | ○ | ○ | ○ |
| Egnyte | ○ | ○ | ○ | ○ | ○ | ○ |
| Nomadesk | ○ | ○ | ○ | ○ | ○ | ○ |
| Evernote | ○ | □ | ○ | ⚡ [2] | ○ | ○ |
| Linkedin Slideshare | ○ | ○ | ○ | ⚡ [4] | ○ | ⚡ [4] |
| JumpShare | ○ | ○ | ○ | ○ | ○ | ○ |

**Table 2.** Summary of the discovered vulnerabilities. Green circles indicate that the service was not found to be vulnerable, blue lines indicate that an HTTP Referer was leaked, but that this was not the direct link to the shared resource, while black lightning bolts indicate that those services were found to be vulnerable to the attack described in this paper. Yellow squares indicate that the service could not be tested because the test was not applicable. Browsers that have green background supports the current Referrer-Policy mentioned in [8] and the ones have red background supports an older draft [3].

[1] Bookmark
[2] PDF
[3] DOC
[4] Any file which can contain a link
[5] Fixed vulnerability in response to our vulnerability disclosure. Hence, these services are, at the time of writing, no longer vulnerable.

documents such as Resume, Book, Academic Journal, Presentation, Poster, etc. All of these mentioned documents are created and rendered as a PDF. The results of our experiments on Overleaf show that it leaks the referrer if the file type is PDF (which is how all compiled LaTeX files are displayed on this service) and the *secret URL* is opened in Microsoft Edge or iOS Safari.

From [3], it can be seen that Edge and Mobile Safari only support an older draft of Referrer-Policy where there are four types as: `"never", "always", "origin" and "default"`. Further investigation showed that Overleaf sets its Referrer Policy to `"origin-when-cross-origin"`. However, this value is not supported by the older draft and hence by Edge and Mobile Safari. The specific policy that they use sends a full URL, stripped for use as referrer [8], when a same-origin request is made but sends an ASCII serialized referrer when making cross-origin requests. Therefore, Overleaf is not vulnerable to the attack scenario we proposed for browsers which support current Referrer-Policy. However, Edge and Mobile Safari do not understand Overleaf's specific policy and this causes a *secret URL* leak. After we informed Overleaf about their security issue, they changed their policy as `"no-referrer"` and added `rel="noreferrer noopener"` to the link tags. Edge and iOS Safari support `rel="noreferrer"` [2]. Therefore, this technique fixed Overleaf's security issue for the browsers in question.

### 4.3 Onehub

Onehub allows a user to create or upload files of types "document", "presentation", "spreadsheet", and "drawing". Among these file formats, only the drawing type does not accept embedding a link in it. By applying our approach to Onehub, we observed that it leaked referrers for all file formats that can contain a clickable link on all of the browsers that we tested.

By using developer tools of Google Chrome, we found that Onehub does not set the Referrer-Policy, which therefore reverts to `"no-referrer-when-downgrade"` which is the default fallback policy if Referrer-Policy is not set according to the W3C standards. This specific policy sends the full referrer from a TLS-protected environment to another TLS-protected environment(potentially trustworthy) regardless of the origin. Since our embedded URL is also a potentially trustworthy URL, Onehub sent the referrer pointing to the *secret URL*. For this case, we conclude that the provider is not aware of this vulnerability since it is not considered or mitigated in any way.

### 4.4 Box

Box is another collaboration service that offers "create" or "upload" operations for "note", "bookmark", "word document", "presentation", "excel document" etc. Bookmarks in Box are nothing but URLs that can also be shared via a secret

link. Box only leaked referrers for bookmark type on Chrome, Edge, Firefox and Mac Safari. However, for iOS Safari and Android Chrome, it leaked the referrer for "bookmarks", "pdfs", and "docs". The reason that mobile browsers and desktop browsers differ in behavior on Box is that Box uses Mozilla's PDF.js library [5] to render content on desktop browsers, whereas it relies on native PDF capabilities on mobile systems. PDF.js seems to remove referrers when generating requests whereas the mobile-native embedded PDF viewers do not. Doc file type is also displayed with JavaScript on a desktop browser so their referrers are not leaked either. Bookmarks are not rendered by Box. Thus, only bookmark type leaks referrers for desktop browsers. While the leak of a bookmark might not have severe security impacts, the leak nonetheless conforms to our threat model. However, on mobile browsers "pdf" and "doc" type files are not displayed by using JavaScript so the referrers are not trimmed by JavaScript. Box embraces `"no-referrer-when-downgrade"` as Referrer-Policy. Without the JavaScript trimming the referrer, the referrer gets leaked due to the unsafe nature of this specific policy. Therefore, shared content on Box is vulnerable when visited with iOS Safari and Android Chrome.

### 4.5 Linkedin Slideshare

Linkedin Slideshare is a platform to share presentations and infographics. Slideshare allows users to upload any type of document. We tested our attack scenario by uploading ".pdf" and ".doc" type files. Slideshare also does not set a Referrer Policy and hence browsers fall back to enforcing `"no-referrer-when-downgrade"`. However, Slideshare renders all uploaded files as a jpeg on desktop browsers regardless of the file type uploaded. Therefore, we could not click the embedded links on Google Chrome, Microsoft Edge, Mozilla Firefox and Apple Safari. On mobile browsers, on the other hand. the documents were natively rendered, making the links clickable for mobile browsers. Our attack scenario succeeded for both document types on iOS Safari and Android Chrome since the Referrer-Policy that is used by Slideshare allows the referrer to be leaked.

### 4.6 Evernote

Evernote is essentially a note-taking application but it allows users to upload several file types. By uploading multiple file types, we concluded that Evernote only leaks the *secret URL* on iOS Safari for PDF file types. On four browsers (Firefox, Chrome, Apple Safari, Android Chrome) Evernote directly downloads the file and opens it with the system default PDF reader. Since this application is separate from the Web browser, no referrer is included when a user clicks on links in the document. We could not test Microsoft Edge since visiting Evernote with this browser returns an error that the browser is not currently supported. On iOS Safari, PDFs are displayed using a built-in PDF viewer. Since Evernote does not set a Referrer Policy, the HTTP Referer is leaked on this browser.

### 4.7 Flock

Flock defines itself as a team communication and collaboration tool that offers instant messaging, group chat, virtual meetings, productivity apps, etc. The testing process for Flock was slightly different than other services. Different types of files are sent through a chat between the first test user and second test user. The recipient of the files then opened the file and clicked the embedded link. On iOS Safari a referrer is leaked and visiting the leaked referrer gave the original document. Thus, we concluded that Flock is vulnerable on iOS Safari.

### 4.8 Other Referrer Leaks

In this section, we describe the four services for which we observed full URLs being leaked as HTTP Referers, but for which accessing the link contained in the referrer did not provide access to the original resource. This shows that the additional step of confirming the vulnerability is necessary, as the mere presence of a referrer in an HTTP request is not a sufficient indicator of this kind of vulnerability.

ShareFile is a file hosting provider which also leaks a suspicious referrer. However, visiting the leaked URL does not give the resource file, it results in an error page that says the file cannot be found. Since a supposed attacker would not be able to open the resource file because the leaked referrer does not refer to the original file, this case is considered not vulnerable.

Hangouts, Fleep, and Flock are instant messaging applications. From Table 2, it can be observed that Hangouts leaks the referrer on three browsers, Flock leaks on Edge, iOS Safari, Apple Safari and Firefox whereas Fleep only leaks in Edge. The referrers leaked by the mentioned instant messaging services contained suspicious links (ex: contains chat IDs) but visiting these referrers only displayed the login page of the web service except Flock on iOS Safari. Since the referrer is not enough to retrieve the original source, these cases are also not considered vulnerable.

### 4.9 Responsible Disclosure

We informed the seven services about the issues that we found during our evaluation. We received feedback from Overleaf in which they said the issue is fixed. In their communications, Overleaf stated that they added `"rel=noreferrer noopener"` to the link tags. Edge supports `"rel=noreferrer"` [3], even though it does not support the Referrer-Policy: `"origin-when-cross-origin"` header. And the `"noopener"` is to prevent a separate issue for target _blank links [4]. From Onehub, we got an email claiming that they have filed an internal ticket to address this issue during their next development cycle. Prior to our full evaluation, we also found Google Docs to be vulnerable. Google fixed the problem via `rel="noreferrer"` and awarded us a bug bounty.

## 5 Countermeasures

To prevent the leaking of sensitive data through referrers, the problem can be approached from both the user and the provider perspective. On the provider's side, a simple solution would be to trim path information in HTTP Referers to only display the hostname as Barth et al. [16] suggested. Alternatively, using the `"rel=noreferrer"` relation for hyperlinks will strip referrers from HTTP requests that result from following such links. This technique is already used by some services (e.g., Google Docs).

If server and client-side cooperate correctly, then the W3C's Referrer Policy can provide thorough protection against leaking information through HTTP Referers. The current Candidate Recommendation [8] describes nine different policies and their effects on HTTP Referer header values, as discussed in Section 2.3. Six out of those nine policies would defend against the problems identified in this work as they prevent the transmission of full URLs as referrers. While the `unsafe-url` policy is explicitly unsafe, the default policy (`no-referrer-when-downgrade`) is unfortunately not secure either. What complicates the use and deployment of the Referrer Policy further is that it requires coordination between the server and the client side. That is, browsers need to implement the necessary logic to understand and implement the Referrer Policy chosen by the server. As our evaluation shows in consensus with online resources (e.g., [3]), Referrer Policy support is unfortunately not thoroughly implemented in all popular web browsers. As such, until all web browsers implement full support for the Referrer Policy, it seems prudent to combine this technique with the link relation attribute discussed above. An alternative way to prevent referrer leaks is deployed by Dropbox where all links in hosted documents are rewritten to go through a sanitation step first. That is, when a link inside a document is clicked, it is first redirected to a referrer sanitation URL [2] to handle the HTTP Referer and then to the desired website.

On the client side, there are several solutions that can prevent referrer leaks. For example, configuration options, such as `network.http.sendRefererHeader`, or `network.http.referer.defaultPolicy` control whether referrer headers should be included at all, or what default Referrer Policy should be applied. Clearly, disabling all referrers would prevent the problems described in this paper, but might be overly aggressive and entail unplanned consequences. Besides configuration settings, browser extensions for Firefox [10] and Chrome [7,6] can conveniently hide the referrer in HTTP Requests. Furthermore, in a managed network setting, such as in an enterprise network, protection techniques can also be applied in a proxy server [23,22]. Finally, modern browsers offer private browsing modes.

For example, Firefox' private browsing mode by default uses a more stringent Referrer Policy (`strict-origin-when-cross-origin`) than the permissive

[2] The endpoint for this sanitation has the representative name of https://www.dropbox.com/referrer_cleansing_redirect

`no-referrer-when-downgrade` policy used in regular mode. Hence, using Firefox' private browsing mode will prevent the attack scenario provided in this paper. Unfortunately, private browsing modes for the other studied browsers do not have such measure regarding Referrer Policy and will leak the referrers.

## 6 Discussion

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

### 6.1 Limitations

The services that do not have a free sign up option could not be included in this paper since they could not be analyzed. This study is conducted on six different web browsers, therefore the discovered vulnerabilities might be a lower bound of the ones that exist in the wild. We believe that we covered the most used browsers, but it is possible that other, less popular ones also do not implement the Referrer Policy properly.

### 6.2 Future Work

As for future work, trying different browsers and other services that are not investigated in this paper can be considered. Investigating whether this vulnerability is known by attackers and how this vulnerability is exploited, could be another further step of this paper. To this end, we could embed links to several websites into our files and follow the same methodology described in this paper. Getting a connection back from any of the websites will imply that someone is exploiting this vulnerability in the wild.

The files can be filled with fake sensitive information and we could examine the use of this information by attackers. Financial data, identity information and online account information are possible intriguing input to be included, similarly to what was done by previous work [27,31].

## 7 Related Work

Several studies have been conducted on the subject of security of file sharing via a link and referrer leaks. However, this is the first paper looking at security threats caused by HTTP Referer leaks on online collaboration services.

A number of papers studied sharing resources via URLs. Ibosiola et al. [18] studied the streaming cyberlocker ecosystem by exploring the streaming links to

video contents. Lauinger et al. [25] and Jelveh et al. [21] investigated copyright infringing content available via unique download links. Antoniades et al. studied service architectures and content of one-click host services [13]. Moreover, Nikiforakis et al. [30] analyzed referrer anonymizing services and showed that RASs leak referrer data to advertisement companies. Studies on security and privacy of online services have been conducted by Balduzzi et al. [15] and Wondracek et al. [33], who investigated the impact of social networks on the privacy of users. The Referrer Policy is presented and explained thoroughly by Dolnak in [17]. Lavrenos and Melon examined popular websites according to Alexa's top one million list and found that the Referrer-Policy is scarcely implemented such that only 0.05% of HTTP responses, and 0.33% of HTTPS responses, contain some form of a valid Referrer-Policy [26]. Andersdotter and Jenden-Urstad analyzed the websites of Swedish municipalities investigating data protection measures according to a number of criteria including the Referrer-Policy [11]. Argyriou et al. [14]. mentioned a possible attack scenario similar to the one we described where the attacker manipulates the embedded URI while investigating OAuth 2.0 Framework. In their study of CSRF attacks Li et al. stated that setting Referrer Policy to accordingly, one can prevent user agents (UA) to suppress the referrer header in HTTP requests that originate from HTTPS domains, preventing the UA from omitting this header by default [28]. Using HTTP Referers are also analysed as a cloaking technique in [20,32,34].

## 8 Conclusion

In this paper, we analyzed 21 different online collaboration services with uploading different types of documents containing a link referring to our servers. The results show that while most of the providers did not leak referrers, the ones that do have a high customer profile and are widely-used. We found seven services that are vulnerable to referrer leaks, and that that referrer leaks can be due to improper use of the Referrer Policy by online services, as well as to limited support to such policy offered by web browsers. We then analyzed the mitigations adopted by online services to prevent referrer leaks.

## Acknowledgements

This work was partially funded by the Office of Naval Research under grants N00014-17-1-2541 and N00014-17-1-2011. We would like to thank the anonymous reviewers for their insightful feedback which helped us improve the final version of our paper.

## References


1. Alexa top lists. https://www.alexa.com/topsites/category/Top/Computers/Internet/On_the_Web/Web_Applications/Storage. Accessed: 2019-02-09.
2. Can i use support tables for html5, css3, etc.
3. caniuse.com rel-noreferrer. https://caniuse.com/#feat=rel-noreferrer. Accessed: 2019-02-09.
4. mathiasbynens.github.io rel-noopener. https://mathiasbynens.github.io/rel-noopener/. Accessed: 2019-02-09.
5. PDF.js. https://mozilla.github.io/pdf.js/. Accessed: 2019-02-09.
6. Referer control. https://chrome.google.com/webstore/detail/referer-control/hnkcfpcejkafcihlgbojoidoihckciin. Accessed: 2019-02-09.
7. Scriptsafe. https://chrome.google.com/webstore/detail/scriptsafe/oiigbmnaadbkfbmpbfijlflahbdbdgdf. Accessed: 2019-02-09.
8. W3C Candidate Recommendation referrer policy. https://www.w3.org/TR/referrer-policy/. Accessed: 2019-02-09.
9. WHATWG link type. https://html.spec.whatwg.org/multipage/links.html#link-type-noreferrer. Accessed: 2019-02-09.
10. Referer control by keepa.com. https://addons.mozilla.org/en-US/firefox/addon/referercontrol/, Mar 2017. Accessed: 2019-02-09.
11. A. Andersdotter and A. Jensen-Urstad. Evaluating websites and their adherence to data protection principles: Tools and experiences. *Privacy and Identity Management. Facing up to Next Steps IFIP Advances in Information and Communication Technology*, page 39–51, 2016.
12. I. Antonellis, H. Garcia-Molina, and J. Karim. Tagging with Queries: How and Why? In *ACM International Conference on Web Search and Data Mining (WSDM)*, page 4, Barcelona, Spain, Feb. 2009.
13. D. Antoniades, E. P. Markatos, and C. Dovrolis. One-click hosting services: a file-sharing hideout. In *ACM SIGCOMM Internet measurement conference (IMC)*, page 223, Chicago, Illinois, USA, 2009. ACM Press.
14. M. Argyriou, N. Dragoni, and A. Spognardi. Security flows in oauth 2.0 framework: A case study. *Lecture Notes in Computer Science Computer Safety, Reliability, and Security*, page 396–406, 2017.
15. M. Balduzzi, C. Platzer, T. Holz, E. Kirda, D. Balzarotti, and C. Kruegel. Abusing Social Networks for Automated User Profiling. In *Recent Advances in Intrusion Detection (RAID)*, volume 6307, pages 422–441, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
16. A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In *ACM conference on Computer and communications security (CCS)*, page 75, Alexandria, Virginia, USA, 2008. ACM Press.
17. I. Dolnak. Implementation of referrer policy in order to control http referer header privacy. *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2017.
18. D. Ibosiola, B. Steer, A. Garcia-Recuero, G. Stringhini, S. Uhlig, and G. Tyson. Movie Pirates of the Caribbean: Exploring Illegal Streaming Cyberlockers. In *International AAAI Conference on Web and Social Media (ICWSM)*, page 10, Stanford, CA, 2018.
19. IETF Network Working Group. Hypertext transfer protocol – http/1.1. https://tools.ietf.org/html/rfc2616#page-140.

20. L. Invernizzi, K. Thomas, A. Kapravelos, O. Comanescu, J.-M. Picod, and E. Bursztein. Cloak of visibility: Detecting when machines browse a different web. *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.
21. Z. Jelveh and K. Ross. Profiting from filesharing: A measurement study of economic incentives in cyberlockers. In *IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 57–62, Tarragona, Spain, Sept. 2012. IEEE.
22. B. Krishnamurthy and C. E. Wills. Cat and mouse: content delivery tradeoffs in web access. In *International conference on World Wide Web (WWW)*, page 337, Edinburgh, Scotland, 2006. ACM Press.
23. B. Krishnamurthy and C. E. Wills. Generating a privacy footprint on the internet. In *ACM SIGCOMM on Internet measurement (IMC)*, page 65, Rio de Janeriro, Brazil, 2006. ACM Press.
24. N. Kushmerick, J. Mckee, and F. Toolan. Towards zero-input personalization: Referrer-based page prediction. *Lecture Notes in Computer Science Adaptive Hypermedia and Adaptive Web-Based Systems*, page 133–143, 2000.
25. T. Lauinger, K. Onarlioglu, A. Chaabane, E. Kirda, W. Robertson, and M. A. Kaafar. Holiday Pictures or Blockbuster Movies? Insights into Copyright Infringement in User Uploads to One-Click File Hosters. In *Research in Attacks, Intrusions, and Defenses (RAID)*, volume 8145, pages 369–389, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
26. A. Lavrenovs and F. J. R. Melon. Http security headers analysis of top one million websites. *2018 10th International Conference on Cyber Conflict (CyCon)*, 2018.
27. M. Lazarov, J. Onaolapo, and G. Stringhini. Honey Sheets: What Happens To Leaked Google Spreadsheets? In *Proceedings of the 9th USENIX Conference on Cyber Security Experimentation and Test (CSET'16)*, page 8, Austin, TX, 2016.
28. W. Li, C. J. Mitchell, and T. Chen. Mitigating csrf attacks on oauth 2.0 systems. *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, 2018.
29. N. Nikiforakis, M. Balduzzi, S. V. Acker, W. Joosen, and D. Balzarotti. Exposing the Lack of Privacy in File Hosting Services. In *USENIX conference on Large-scale exploits and emergent threats*, page 8, Mar. 2011.
30. N. Nikiforakis, S. Van Acker, F. Piessens, and W. Joosen. Exploring the Ecosystem of Referrer-Anonymizing Services. In *Proceedings of the 12th international conference on Privacy Enhancing Technologies (PETS'12 )*, volume 7384, pages 259–278, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
31. J. Onaolapo, M. Lazarov, and G. Stringhini. Master of Sheets: A Tale of Compromised Cloud Documents. In *Proceedings of the Workshop on Attackers and Cyber-Crime Operations (WACCO)*, Goteborg, Sweden, 2019.
32. D. Y. Wang, S. Savage, and G. M. Voelker. Cloak and dagger. *Proceedings of the 18th ACM conference on Computer and communications security - CCS 11*, 2011.
33. G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A Practical Attack to Deanonymize Social Network Users. In *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2010. IEEE.
34. B. Wu and B. D. Davison. Detecting semantic cloaking on the web. *Proceedings of the 15th international conference on World Wide Web - WWW 06*, 2006.
35. G. Zheng and S. Peltsverger. Web analytics overview. *Encyclopedia of Information Science and Technology, Third Edition*, page 7674–7683.