Leakuidator: Leaky Resource Attacks and Countermeasures

Mojtaba Zaheri(✉) and Reza Curtmola

New Jersey Institute of Technology, Newark, NJ 07102, USA
{mojtaba.zaheri,reza.curtmola}@njit.edu

Abstract. Leaky resource attacks leverage the popularity of resource-sharing services to conduct targeted deanonymization on the web. They are simple to execute because many resource-sharing services are inherently vulnerable due to the trade-offs made between security and functionality. Even though previous work has shown that such attacks can lead to serious privacy threats, defending against this threat is an area that has remained largely unaddressed.

In this work, we advance the state of the art on leaky resource attacks on both attack effectiveness and attack mitigation fronts. We first show that leaky resource attacks have a larger attack surface than what was previously believed, by showing reliable attack implementations that work across a broader range of browsers and by identifying new variants of the attack. We then propose `Leakuidator`, the first client-side defense that can be deployed right away, without buy-in from browser vendors and website owners. At a high level, `Leakuidator` identifies potentially suspicious requests made when a webpage is rendered and for each such request: (1) renders the request by first removing cookies from it, and (2) initiates a second request that is identical with the original request (i.e., contains the cookies that were removed), but does not render its response. This additional request maintains compatibility with existing web functionality, such as analytics and tracking services. We have implemented `Leakuidator` as a browser extension for three Chromium-based browsers. Experimental results show that `Leakuidator` introduces a small overhead and thus the impact on user experience is minimal. The extension also includes usability knobs, allowing users to reuse past choices and to adjust how strict is the criteria for identifying potentially suspicious requests.

1 Introduction

The ability to privately share digital media resources, such as image, video, and audio files, provides a convenient mechanism to socialize and interact online, as shown by the popularity of websites that provide resource sharing services among their users [2,5,6]. Unfortunately, resource-sharing services were also shown to introduce avenues to conduct serious privacy violation attacks [28,30]: Information leaked by these services allows an attacker to infer the identity of a victim that visits an attacker-controlled website, which would not be possible otherwise.

J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 399, pp. 143–163, 2021.
https://doi.org/10.1007/978-3-030-90022-9_8

The first instantiation of the attack was the *leaky image attack*, introduced in a seminal work by Staicu and Pradel [28], using an image as the shared resource. The attacker first shares privately an image with the target victim using an image sharing service. The attacker then embeds a link to the shared image on a webpage she controls. When a visitor loads that webpage, the image will be successfully retrieved only if the visitor is the targeted victim, since only the victim is allowed to retrieve the image (assuming the victim's browser is logged into the image sharing service). By observing the success of loading the image, the attacker will know if the intended victim has visited the attacker's website.

Even though previous work introduced the attack using images (*i.e.*, leaky images [28]), we found that the attack works with any resource that can be privately shared with the victim and can be rendered on a webpage. In particular, the attack also works with other media files, such as video and audio files. Thus, we generically refer to the attack as a *leaky resource attack*.

The leaky resource attack is a targeted privacy attack, in which an individual browsing an attacker-controlled webpage can be uniquely identified. This is in contrast with other known de-anonymization techniques, such as third-party tracking [15] (*e.g.*, tracking pixels or tracking IPs) or social media fingerprinting, that do not provide this level of accuracy. As such, leaky resources can be abused in a variety of privacy-sensitive scenarios [28], including law enforcement gathering evidence regarding the online activity of individuals, oppressive governments tracking political dissidents, de-anonymizing reviewers for a conference paper, blackmailing individuals based on their online activity, or health insurance companies discriminating individuals based on their online activity.

Although the most natural way to implement a leaky resource attack is to rely on JavaScript, in this work we focus on the scriptless version which uses only HTML. A scriptless attack is more challenging to execute successfully, but will affect even privacy-aware users who limit or disable scripts in their browser [4,26].

In this paper, we advance the state of the art in leaky resource attacks for both attack effectiveness and attack mitigation. We first show that the attack surface is larger than was previously believed, by identifying three new variants of the attack. In the first variant, we show that scriptless leaky resource attacks based on `video` and `audio` HTML tags provide increased reliability compared to the previously known `object` HTML tag: The attack now works against all the vulnerable services we identified and across all browsers we tested with (Firefox, Edge, Chrome). By testing the attack on different categories of sharing services, we reveal a concerning reality: 17 popular resource sharing services are still vulnerable despite the attack being known for more than a year (12 of these services are newly found to be vulnerable by our work).

The other two attack variants work for websites that integrate with a file hosting website and offer access to the hosted resources. When a resource is shared on the file hosting website, this sharing is inherited on the integrator website. Even if the file hosting website fixes a leaky resource vulnerability,

the attack may still be possible by accessing the shared resource through the integrator website (attack variant two). Finally, we show a de-anonymization attack that can be performed non-interactively, *i.e.*, without even involving the victim (attack variant three).

On the attack mitigation front, we propose `Leakuidator`, the first client-side defense that can be deployed right away, without buy-in from browser vendors and website owners. As part of the defense, the browser renders each webpage by removing cookies from potentially suspicions requests made by that webpage before sending those requests. To maintain compatibility with analytics and tracking services, `Leakuidator` initiates for each original request that was deemed potentially suspicious a second request that is identical with the original request (i.e., it contains the cookies that were removed). However, the response to this second request is not rendered. After loading the webpage, if any observable differences are detected between the two responses, the original request is deemed dangerous and the user gets a small notification on the browser toolbar; if the user deems the webpage trustworthy, she can reload it with the defense disabled. `Leakuidator` also includes usability knobs, allowing the user to reuse past choices and to adjust the criteria for identifying potentially suspicious requests.

To realize this seemingly simple, yet elegant solution, several challenges need to be overcome (Sect. 5.1). We have implemented `Leakuidator` as a browser extension for three Chromium-based browsers (Chrome, Edge, and Brave). Our experimental results show that `Leakuidator` introduces a small overhead, thus minimally impacting user experience. For example, the majority of websites tested experience less than 200 ms in additional load time when `Leakuidator` is active.

1.1 Motivating Examples

We now describe deanonymization scenarios that leverage leaky resource attacks.

Law Enforcement. Given a target suspect, law enforcement agencies can use this attack to learn if the suspect is visiting specific websites. Also, they can use this attack to link an anonymous account in one resource sharing service to a known user account in another resource sharing service, hence deanonymizing the suspect. For example, FBI employed *Network Investigative Techniques* with similar deanonymization goals since at least 2002 [7].

Sensitive Websites. Discreet or extramarital affairs, users of pornography websites, and sextortion are targets for this attack, as users tend to keep their identities anonymous in these situations. For example, FBI booby-trapped a video to catch a suspected Tor child sextortionist via Dropbox [3].

Blind Peer Review. Academic conferences and journals rely on a blind peer review process, in which the identity of the reviewers is hidden from the authors. Using this attack and starting with the list of program committee members (or editorial board), the authors of an article can mount a series of attacks that leads to learning the identity of the article reviewers [30].

Journalism and Critics. Leaky resource attacks can reveal the identities of anonymous government critics by targeting their social media accounts. For example, when a targeted critic visits an attack webpage with links to leaky resources for a list of candidate accounts, a successful request to one of the links may reveal the critic's identity. Another way to deanonymize such critics is to use the account linking strategy, trying to link the target anonymous account to a known account. As an example, the U.S. government demanded that Twitter release information to identify an account holder whose tweets have been critical of the Trump administration's immigration policies [10].

Insurance Coverage. Insurance companies can use information about the websites visited by users to find their concerns about specific disease, thus affecting the companies' decisions about the coverage and premiums.

2 Background on Leaky Resource Attacks

This section overviews previous work on leaky resource attacks. The leaky resource attack was introduced in the context of a leaky image [28]. At a high-level, the attack consists of the following three steps:

1. Setting up the Attack: The attacker uploads an image to an image sharing service. The attacker then privately shares the image with the victim. The image sharing service uses cookies for user authentication and allows access to the shared image via a state-dependent URL (SD-URL). An SD-URL is a URL for which the response to a user request for the URL is different depending on the user's state with respect to the sharing service. For example, if the user is the targeted victim, the resource will be loaded, otherwise it will not be loaded.

To determine if the victim is visiting an attack webpage P, the attacker embeds in P an SD-URL for the shared image. P is controlled by the attacker (or, the attacker is able to embed in P a request for the SD-URL of the shared image).

2. Luring the Victim: The attacker lures the victim to visit the attack page P.

3. Communicating the Attack Results: When the victim loads the attack page P, the victim's browser makes a cross-origin request for the shared image at the SD-URL. The Same-Origin Policy would normally prevent the attacker from reading the contents of the cross-origin response. However, the attacker can bypass this policy using an XS-leak [30] to learn information about the response: If the image is successfully retrieved, the attacker is certain that the targeted victim is visiting the attack page.

Scriptless Attack. Previous work has shown a JavaScript-based version of the attack [28] (included in Appendix A for reference). However, privacy-aware users may disable JavaScript or use a protection mechanism that prevents JavaScript-based XS-leaks. Even under such an environment, the attack can still be implemented using only HTML, without relying on JavaScript or CSS. The attacker

uses an HTML tag that allows to load fallback content in case the primary content fails to load. This fallback-based mechanism can be used to simulate a *if-then-else* control flow instruction in pure HTML. Previous work identifies the `object` HTML tag to achieve this functionality [28], as shown in Fig. 1.

```
<object data="State-Dependent-URL" type="image/png">
  <object data="Fallback-URL" type="image/png"></object>
</object>
```

Fig. 1. Communication method using the `object` HTML tag. If the outer `object` element (State-Dependent-URL) fails to load, then the fallback is to load the inner `object` element (Fallback-URL, controlled by the attacker).

Other Attack Scenarios. Previous work has shown how to extend this attack from identifying a single user to identifying any specific user in a group of users [28]. This can be done efficiently for a group of n users by embedding $log(n)$ SD-URLs in the attack page and by sharing specific groups of the $log(n)$ resources with each targeted user in the group.

Another attack scenario seeks to link multiple identities a single individual has in different sharing services. For example, an attacker wants to know whether two user accounts in two different sharing services belong to same individual. The attacker shares a resource in each sharing service with the victim and then embeds the SD-URLs of these two resources in an attacker-controlled webpage. Successful response to both requests confirms that the two accounts belong to the same person, potentially linking a known account to an anonymous account.

3 Threat Model and Security Objectives

We consider attackers that can bring together the following necessary ingredients for a successful leaky resource attack:

1. The attacker and the victim are users of the same resource sharing service.
2. The resource sharing service allows its users to share resources privately with each other and authenticates users through cookies.
3. The attacker convinces the victim to visit the attack page (which is controlled by the attacker) while the victim is logged into her account with the resource sharing service (which is not controlled by the attacker).
4. The attacker can determine if the victim loaded the resources successfully.

The attack is effective because these requirements can be achieved in multiple ways and are within easy reach of the attacker. For requirement #1, resource sharing services are very popular, so the victim may have an account; also, many such services have free membership and the attacker can just create an account with a service used by the victim. For requirement #2, these are the de facto

mechanisms for many resource sharing services. Some popular services, such as Google Drive, even offer the option to not notify the target users when a file is shared with them. Requirement #3 can be achieved in multiple ways, including via phishing emails, or via a watering-hole approach [30]. It is common for many users to be logged into popular services while surfing the Internet.

Requirement #4 is crucial for the attack and can be achieved as follows. The attack page contains a state-dependent URL (SD-URL) [30] that points to content on the target website (*i.e.*, the resource sharing service). When a user makes a request for the SD-URL, the response is different depending on the user's state with respect to the target website: If the user is the targeted victim, the content will be loaded, otherwise it will not be loaded. The attacker can learn information about this response based on a *XS-leak* [30] that bypasses the Same-Origin Policy which normally prevents the attacker from reading a cross-origin response.

Security Objectives. To properly mitigate leaky resource attacks, we envision that the following security objectives need to be achieved:

SO1: Prevent Leaky Resource Attacks. Suspicious third party requests that may lead to leaky resource attacks should be accurately detected and prevented.
SO2: Allow End Users to Control their Privacy Level. Users should have the ability to decide whether a suspicious request is dangerous and violates their privacy or not, and therefore have control on their privacy.
SO3: Limited Impact on Existing Web Functionality. A defense should not interfere with existing web functionality, including third party authentication, personalized advertisement, tracking, and analytics.

4 Attacks

We present three new variants of the leaky resource attack.

4.1 Variant 1 Attacks: State Dependent URLs (SD-URLs)

Variant 1 introduces two new HTML-only XS-leaks to communicate the attack outcome to the attacker, using the `audio` and `video` HTML tags. Our testing results described in Sect. 4.4 show that these are more reliable than previously known XS-leaks that rely on the `object` HTML tag [28]. Unlike the `object` tag, the new XS-leaks work reliably across different browsers, and they work for some resource sharing websites for which the `object` tag does not seem to be effective.

Another drawback of using the `object` tag as an XS-leak mechanism is that websites can use the X-Frame-Options HTTP response header to prevent a browser from rendering an image in an `object` tag. The new XS-leaks we introduce are not subject to this limitation.

Communication Method: Video Tag. Multiple source elements can be used inside a `video` HTML tag to cover browsers that support different video types. Normally, this is used by website authors to specify multiple alternative media resources for media elements. However, Fig. 2 shows that alternatives can be used to trigger a fallback behavior that mimics an *if-then-else* control flow. Both resources we used have the type webm, but other video file types can be used as well. The attacker can ensure the specific file type is supported by the browser by checking the HTTP Request Headers and preparing an appropriate webpage.

Communication Method: Audio Tag. Similar to the `video` tag, multiple source elements can be put inside an `audio` tag to cover browsers that support different audio types. Figure 3 shows how it is used to trigger a fallback behavior. Both resources used have the type ogg, but other audio file types can be used.

```
<video>
    <source src="State-Dependent-URL" type="video/webm">
    <source src="Fallback-URL" type="video/webm">
</video>
```

Fig. 2. Communication Method: Video HTML Tag. If the first source (State-Dependent-URL) cannot be loaded, then the fallback is to load the second source (Fallback-URL, controlled by the attacker).

```
<audio>
    <source src="State-Dependent-URL" type="audio/ogg">
    <source src="Fallback-URL" type="audio/ogg">
</audio>
```

Fig. 3. Communication method: audio HTML tag.

4.2 Variant 2 Attacks: Mediated SD-URLs

Integrators of sharing services can add extra features that are not available in the sharing service itself. For example, a website can manage resources from multiple cloud accounts in one place by integrating with multiple cloud services. As another example, an automated software testing or delivery service can integrate with source code hosting services. A common property of integrator services is that they often offer access to the resources shared in the sharing service through different URLs on the integrator's domain. If such URLs exhibit the state dependent property, we refer to them as mediated SD-URLs.

For example, the attacker shares *resource* with the victim on a sharing service *sharingX.com*, and the URL to access *resource* is *sharingX.com/resource*. If the victim has an account with a website *integrator.com* that integrates the *sharingX* service and offers access to *resource* through a mediated SD-URL on

the integrator's site (e.g., *integrator.com/sharingX/resource*), then the mediated SD-URL can be used to conduct leaky resource attacks. We identified mediated SD-URLs in Buddy, a CI/CD system integrating with GitHub, GitLab, and Bitbucket.

The relevance of this attack variant is that even if the sharing service deploys a fix for leaky resource attacks on its own domain, the problem may persist through its integrators, which are not necessarily following the same security practices. For example, the sharing service may start using a more restrictive policy using the *SameSite* cookie attribute, but this does not guarantee that its integrators will follow similar policies.

4.3 Variant 3 Attacks: No Victim Interaction

We found that it is possible to link user identities without the need for the victim to actually interact or load any webpage. This attack variant also involves an integrator that integrates with a sharing service.

We present a working example using the Slack integration of Google Drive. The attacker's goal is to link two account IDs, one for Google Drive and one for Slack. If the attacker knows the real identity behind one of these IDs, she can then deanonymize the other account ID. The attacker shares a resource privately with the victim in Google Drive. The attacker then uses the Google Drive integration in Slack to send this shared resource to a set of target Slack users, e.g., by attaching the resource to a message to the targeted users. The Google Drive bot in the attacker's Slack message box will prompt to give permission to all the target Slack users except for one. That one user already has permission to access the resource, and the attacker will be able to successfully link the victim's Google Drive and Slack account IDs without even interacting with the victim.

4.4 Attack Test Results

Overall, we found 17 popular sharing services that are vulnerable to the leaky resource attack, as shown in Table 1. Among these, we informed 6 services through the HackerOne and BugCrowd programs and we anonymized them as SharingService1 through SharingService6 in order to comply with their disclosure policies. We found that 5 out of 8 sharing services reported to be vulnerable by Staicu and Pradel [28] remain vulnerable to the leaky resource attack, namely Google Drive, SharingService4, One Drive, Skype, and SharingService6. Interestingly, we found that the scriptless attack based on the `object` tag does not work reliably: for SharingService4 it did not work at all, and for two other services (One Drive, Google Drive) it only worked in one out of three browsers we tested with. However, using the `video` and `audio` tags, the attack works unilaterally for these services in all three browsers. Among the remaining 3 services, Facebook made an ad hoc fix to the vulnerable API, Twitter changed the sharing mechanism, and GitHub restricted the cookies using the *SameSite* attribute.

We then examined major sharing services listed on wikipedia for social networking, image sharing, video sharing, file hosting, source code hosting, blog

hosting, and instant messaging, after excluding services that do not have an English language interface, do not have free membership, or do not have the private sharing feature. As a result, we found additional vulnerable services: 4 cloud storage services (SharingService3, CloudMe, IDrive, Koofr) and 6 source code hosting services (SharingService1, SharingService2, Launchpad, Assembla, Buddy, Gitea). Finally, we surveyed several popular photo sharing services, which revealed two additional vulnerable services (SharingService5, Google Photos).

Discussion. We found that the `object` tag XS-leak can be affected by many factors and is unreliable. Although we found SD-URLs in 16 services, using the `object` tag the attack is successful in only 9 services with Firefox, and in only 4 services with Chrome/Edge. In contrast, the new tags we found are more reliable. Out of 16 services tested, the `video` tag led to a successful attack in all

Table 1. Test results for the leaky resource attacks. We denote the browser as follows: E for Microsoft Edge 87.0, F for Mozilla Firefox 83.0, and C for Google Chrome 87.0. The services are grouped into source code hosting (rows 1–6), cloud storage (rows 7–13), photo sharing (rows 15, 16), and others (rows 14, 17). For all services, we used variant 1 of the attack (SD-URL), except for Buddy which is vulnerable to variant 2 (mediated SD-URL). "Not Supported" denotes the service did not support the respective media type. "Not Successful" denotes that the attack was not successful in any of the tested browsers.

#	Service	`object` tag [28]	`video` tag	`audio` tag
1	SharingService1	Not Successful	F	F
2	SharingService2	Not Successful	E, F, C	E, F, C
3	Launchpad	F	F	F
4	Assembla	F	F	F
5	Buddy	F	F	F
6	Gitea	Not Successful	F	F
7	SharingService3	F	F	Not Supported
8	Google Drive	F	E, F, C	E, F, C
9	One Drive	F	E, F, C	E, F, C
10	SharingService4	Not Successful	E, F, C	E, F, C
11	CloudMe	E, C	E, F, C	E, F, C
12	IDrive	Not Successful	E, F, C	E, F, C
13	Koofr	E, C	E, F, C	E, F, C
14	Skype	E, F, C	E, F, C	E, F, C
15	SharingService5	SD-URL Not Found	F	Not Supported
16	Google Photos	F	Not Supported	Not Supported
17	SharingService6	E, F, C	E, F, C	E, F, C

16 services with Firefox, and in 9 services with Chrome/Edge. Out of 14 services tested, the `audio` tag led to a successful attack in all 14 services with Firefox, and in 9 services with Chrome/Edge.

The reason why XS-leaks based on `video` and `audio` tags were not successful in some services when using Chrome/Edge browsers is that different browsers behave inconsistently when the *SameSite* attribute is not set at all. Chromium-based browsers versions 80 and above treat cookies as if a *lax* SameSite attribute is set, whereas Firefox (tested up to version 83) treats them as if SameSite is set to *none*. We also note that version 80 of Chromium-based browsers was rolled out on August 11, 2020, meaning more services are vulnerable when using these browsers with version prior to 80 (*e.g.*., all 16 services tested using video tags and all 14 services tested using audio tags are vulnerable).

4.5 Responsible Disclosure and Feedback from Affected Services

We summarize the responses received after performing responsible disclosures.

Google confirmed the issue for its Drive and Photos products, placing it in a larger context of XS-Search attacks. In response, Google is considering several larger scale changes to their products: a) Auditing which search endpoints exist in their web services that need to be protected against, b) Experimenting with different defenses that do not break existing user functionality but also are effective, c) Working with web browsers to find defenses.

SharingService3 is a top e-commerce and media sharing service. They acknowledged the vulnerability and issued a bug bounty.

Microsoft's response suggests this is as an acceptable risk for its OneDrive and Skype products, as their main focus is to ensure the integrity, availability, and confidentiality of these services. They seem to discard the privacy implications of the attack, without providing any clear justification.

SharingService2 is a major source code hosting service and acknowledged this as a known issue. They considered the SameSite cookie attribute as a defense, but decided against it because it would break one of the integrated services.

SharingService1 is a major source code hosting service and acknowledged the issue, but considers it an acceptable risk. **SharingService4** is a top cloud hosting service who acknowledged the issue and is working on a feasible solution.

SharingService5 is a top media hosting service and confirmed that their internal team is already aware of this issue.

SharingService6 questioned the effectiveness of the attack because it may be non-trivial to convince a victim to accept an invitation from the attacker. In addition, they acknowledged evaluating the SameSite cookie attribute as a defense is a work in progress because some product features would be affected.

Koofr acknowledged the issue and set the cookie SameSite attribute to *lax* as a fix. **Slack** were informed about the attack variant 3, but did not respond.

Among the six anonymized vulnerable services, it is notable that some have either deployed a fix or are actively working on identifying a practical fix. Hereby we also note the duplicitous response from some of the vulnerable services: On one hand, they downplay these privacy issues, with no plan for deploying a

fix. On the other hand, they asked us specifically not to publicly disclose the vulnerabilities. Such an attitude may suggest that these services may not value the privacy of their users; it may also pinpoint to limitations in the disclosure policies of bug bounty programs.

5 Leakuidator: A Defense Against Leaky Resource Attacks

5.1 A Solution Suggested by Prior Work

The following suggestion was made in prior work to prevent the leaky image attack [28]. The browser should render authenticated image requests only if there are no observable differences between an authenticated request and non-authenticated one. For this, the browser should perform one request with third-party cookies and one without, and only render the image if the responses are equivalent. Although this idea is inspirational, no details are given on how to take this idea from the suggestion stage to that of an actual defense. In fact, significant challenges need to be overcome.

Algorithm 1. Proposed defense against leaky resource attacks

Input: *Req*, the request to be evaluated

1: **if** *Req* contains cookies **and** *Req.source* is not equal to *Req.target* **then**
2: Remove cookies from *Req* and send the request *Req*
3: Deliver to the browser the response *Res* received for *Req*
4: Send another request *Req′* identical to *Req* (i.e., with cookies)
5: Let *Res′* be the response received for *Req′*
6: **if** *Res* is not equal to *Res′* **then** classify the request as dangerous

First, the browser should wait for both responses to arrive in order to evaluate if they are equivalent. This will delay the rendering of the webpage and has detrimental effects to the user experience, not to mention it is not clear what to do if one of the requests gets delayed or lost. Second, blocking the rendering of the webpage until both responses are received requires modifying the browser. Solutions that do not require buy-in from browser vendors are preferable, as that is a challenging and time-consuming process. Finally, blocking the rendering of the requested image when an observable difference is detected will still cause information leakage, because blocking or not blocking of a request is a binary condition that can be used for information leakage attacks.

5.2 Leakuidator: A Defense Against Leaky Resource Attacks

We propose Leakuidator, a defense that applies the steps described in Algorithm 1 to each request made by the browser when loading a webpage. First,

the defense identifies a request as potentially suspicious if the request contains cookies and is cross-origin, i.e., the source and target origins are different (Line 1). The source origin is the origin of the webpage making the request (*i.e.*, the URI shown in the browser's address bar), whereas the target origin is the origin indicated by the request. The next step is to remove the cookies from the request and send the modified request (Line 2). The response to this request will be rendered by the browser in the webpage (Line 3). The defense then makes a second request, identical to the original request, *i.e.*, the request includes the cookies that were removed (Line 4). The responses to the original request and this second request are compared and if any observable differences are detected, the request is deemed dangerous (Line 6).

The proposed approach addresses the aforementioned challenges. The response to the modified request is rendered by the browser as soon as it is received and before being compared to the second response, thus no delays are imposed. `Leakuidator` can be implemented as a browser extension and does not require buy-in from browser vendors and website owners. Finally, the response to the modified request does not rely on cookies and will be the same even if the user is the targeted victim or not, hence preventing additional information leakage.

Impact on Existing Web Functionality. We now analyze the impact of the proposed defense on existing web functionality.

Third Party Authentication. To keep third party authentication unaffected (e.g., single sign-on services), potentially suspicious requests that are followed by a top-level navigation are excluded from the defense. Since such requests are followed by a top-level navigation to another webpage, the communication methods used in a leaky resource attack are deemed impractical, as the attacker webpage will not get the chance to communicate the result of a dangerous request back to attacker after leaving the attack webpage.

Analytics. The second request, which contains cookies, ensures that tracking and analytics services remain unaffected. However, since the first request is sent without a cookie, the tracking server will normally respond with a *set-cookie* header, which may override any existing cookie and user id and history. Thus, we remove the *set-cookie* header from the response to the first request. If observable, this modified response can help an attacker to at most learn if the user has previously visited or logged in to a third party service or not. However, this is not enough to reveal the identity of the target user.

Personalized Advertisement. An advertisement service may send a personalized ad based on the cookies in the second request. This personalized ad will not be rendered in the browser, instead the user may get a generic ad in response to the modified original request. However, the user will be notified by `Leakuidator` and has the option to exclude the ad service from the protection. The user will

then receive personalized ads for subsequent visits to the webpage. Therefore, assuming the user consents, the impact on personalized ads is minimal.

5.3 `Leakuidator` Implementation Details

We have implemented `Leakuidator` as a browser extension for three Chromium-based browsers, Chrome, Edge, and Brave. The extension consists of 1,674 lines of JavaScript and HTML code, amounting to 362 KB, including 36.6 KB for the *punycode* and *publicsuffixlist* libraries, 231 KB for the list of public suffixes, and 22.8 KB for the extension icon. To decide if there are observable differences between the two responses (Line 6 of Algorithm 1), `Leakuidator` considers the *Status*, *Content-Encoding*, *Content-Range*, *Content-Length*, and *Etag* response headers.

The second request initiated by `Leakuidator` (line 4 in Algorithm 1) can be implemented as a HEAD request which is identical to a GET request except that the server only returns the headers in the response. `Leakuidator` can use a HEAD request instead of a GET because the extension only uses information from the response headers to detect potentially dangerous requests. The fact that the message body is not returned reduces `Leakuidator`'s communication overhead.

If the responses to a potentially suspicious request are identical, then `Leakuidator` does not affect the webpage rendering. Otherwise, after the webpage is loaded, the user gets a small notification in the extension icon on the browser toolbar regarding the existence of dangerous requests; if the webpage is deemed trustworthy, the user has the option to reload it with the defense disabled (i.e., cookies will not be removed from the original request).

This notification on the browser toolbar has minimal impact on the user experience. If the user clicks on the extension icon, a popup window presents a list of potentially dangerous requests attempted by the webpage. For each such request, the source and target origins are shown.

To minimize the impact on the user experience, `Leakuidator` includes usability knobs. First, when presented with a list of potentially dangerous requests, users can record their choices for future use (*i.e.*, users will not be notified again for such future occurrences). Specifically, for each potentially dangerous request, the user is given four options:

1. Exclude Pair: Exclude the ordered pair of source and target from the protection and allow the suspicious requests with this pair without modification.
2. Exclude Target: It is similar to *Exclude Pair*, but excludes from the protection all requests to the target origin, regardless of the source.
3. Protect Pair: Always provide protection for requests with this ordered pair of source and target.
4. Protect Target: Similar to *Protect Pair*, but provides protection for all requests to the target origin, regardless of the source.

Second, `Leakuidator` can operate in two modes, allowing users to adjust how strict is the criteria for identifying potentially suspicious requests. In the

Exact mode, the decision if the source and target origins are different (Line 1 of Algorithm 1) is based on comparing the entire URI (*i.e.*, combination of domain, protocol and port altogether). In the *Relaxed* mode, this decision is based only on the registrable part of the URI's domain, as identified by public suffixes maintained at `publicsuffix.org`. We introduced the *Relaxed* mode after noticing that most popular websites make legitimate requests to their subdomains or sibling subdomains. For example, a request from *outlook.live.com* to *onedrive.live.com* is not flagged as suspicious under the *Relaxed* mode, as the registrable parts of these domains is the same, *live.com*.

Additional Implementation Details. The `Leakuidator` browser extension has an options page, allowing the user to change the operating mode and prior decisions. For the mode, the user has two options: *Relaxed* and *Exact*. For prior decisions, users have access to the Exclude and Protect lists, and can edit their prior decisions stored in these lists. The lists are maintained separately for the *Relaxed* and *Exact* modes to avoid any conflicts or confusion by the user.

The requests that trigger a top-level navigation are exempt from mitigation. To identify such requests, `Leakuidator` evaluates *fetch metadata* request headers, by checking that *Sec-Fetch-Mode* and *Sec-Fetch-Dest* request headers are set to *navigate* and *document* respectively.

6 Security Analysis

Even though `Leakuidator` has the potential to provide immediate protection against leaky resource attacks, users may be skeptical to install a new browser extension. To address this, we will make the extension's source code publicly available and leverage public scrutiny to build confidence in it. We will also publish the extension on the Chrome Web Store, whose vetting process and the user review-based rating will provide additional evidence about its trustworthiness.

We now show that `Leakuidator` meets the security objectives presented in Sect. 3.

SO1: Prevent Leaky Resource Attacks. The root cause of a leaky resource attack is the ability of a webpage to infer information about the response to a cross-origin request that contains cookies. `Leakuidator` determines potentially suspicious requests and ensure they are rendered only after removing the cookies from them. This eliminates the root cause of the attack, thus preventing variants 1 and 2 of the attack. We note that variant 3 of the attack cannot be prevented using a pure client-side defense, since it does not involve interaction with the victim. To mitigate attack variant 3, fixes are needed on the server side (*i.e.*, the sharing service and the integrator).

SO2: Allow End Users to Control their Privacy Level. Security-conscientious users may want the ability to maintain control over their privacy level, especially with regard to avoiding leaky resource attacks, regardless of the

various policies and implementations deployed by websites. `Leakuidator` notifies the user whenever it detects a dangerous request, allowing the user to decide if the request is benign or not. `Leakuidator` also allows users to reuse past choices and to control how strict is the criterion for identifying potentially suspicious requests.

SO3: Limited Impact on Existing Web Functionality. As analyzed in Sect. 5.2, we expect `Leakuidator` to have minimal impact on existing web functionality.

7 Experimental Evaluation

Experimental Setup. This section evaluates the overhead introduced by the `Leakuidator` defense. We compared three cases: a *Baseline* case with no extension installed, and *Relaxed* and *Exact* cases with the `Leakuidator` extension installed and set to the *Relaxed* and *Exact* modes, respectively.

We wanted to test popular websites that have a large user base. Thus, we chose the Alexa top 15 English websites at paper submission time. Data points in this section's graphs are averages over 20 independent runs. The graphs also show the standard deviation. In each run, we used the Puppeteer automation framework [8] to launch a new browser instance, load the website, extract the measurement information, and then close the browser instance. A separate browser instance is used each time to make sure the test is not affected by browser caching. Measurements are extracted from the Chrome Developer Tools UI.

We ran experiments using Puppeteer Core 5.4.1 and Chrome 87.0 on a system with Intel Core i7-7820HQ CPU with 8 cores (each running at 2.90 GHz), 16 GB RAM, and Intel HD Graphics 630 (kBL GT2) card, running Ubuntu 18.04.5 LTS, kernel v.5.4.0-59-generic.

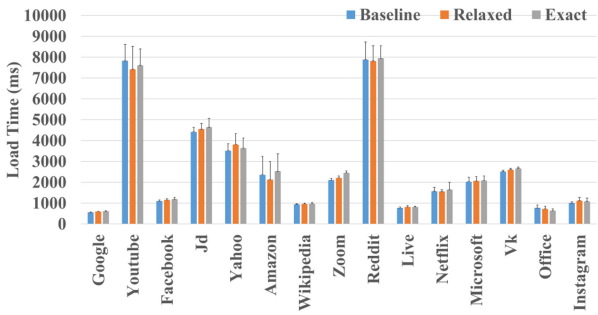

Fig. 4. Load Time for the Baseline, Relaxed, and Exact cases (in milliseconds).

Load Time. Figure 4 shows measurements for the *load time*, defined as the time between when the browser makes the first request to the target website, and when

the website is loaded in the browser. The extension's effect on the load time is limited to the modifications made on the original request, *i.e.*, cookie removal for potentially suspicious requests. The second request initiated by `Leakuidator` does not influence the load time because all responses to requests initiated by the web page are immediately delivered to the webpage without delay. Each website has its own implementation and behaves differently when receiving cookie-less requests. Some websites have identical responses regardless of the presence of cookies. Other websites issue different responses when cookies are removed. The changes in response may increase the load time (*e.g.*, YouTube, Yahoo, Reddit) or decrease it (*e.g.*, Facebook). The majority of websites have slightly faster load time when the extension is not installed. Still, the difference between the three modes is at most 646 ms, with the majority of websites experiencing a difference less than 200 ms. This suggests the impact on user experience is minimal.

Network Traffic. Making two requests instead of one will affect the network traffic. Figures 5 and 6 show the two metrics we used for the network traffic: the number of requests and the amount of data transferred. These are measured for the interval between when the first request is made to the website and when the website finishes loading. When the extension is installed, we add the network traffic caused by the extension during this time interval. Figures 5 and 6 show these measurements. Intuitively, one expects that network traffic should be doubled by having this extension installed, but this is not seen in our measurements. One reason is that third party requests represent a small portion of the requests made by websites. Another reason is that removal of cookies from third party requests will influence the response. For example, the lack of authentication cookies may result in no additional subsequent requests, which are normally sent if cookies were present. Without cookies, some websites return an HTTP error response which is smaller in size than the target resource, whereas other websites may respond with an error webpage that can have larger size than the target resource.

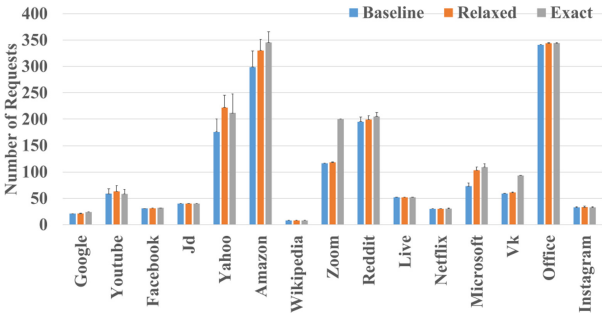

Fig. 5. Number of Requests for the baseline, relaxed, and exact cases.

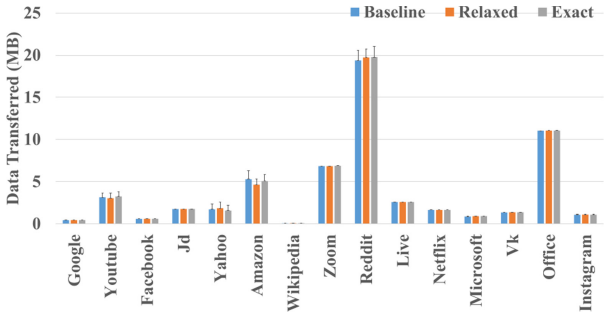

Fig. 6. Data transferred for the baseline, relaxed, and exact cases (in MBs).

Figure 5 reveals that some websites show an increased number of requests due to heavy integration of third party services with requests containing cookies for advertisement and tracking (*e.g.*, Yahoo, Amazon, and Microsoft). However, for the majority of websites, the presence of the extension has minimal impact, adding less than 6% additional requests. Looking at the amount of data transferred, Fig. 6 shows that `Leakuidator` impacts websites differently: the amount increases for Yahoo, but not for Amazon and Microsoft. This shows that `Leakuidator`'s impact depends on the specifics of the third party requests.

When comparing the performance of `Leakuidator`'s *Relaxed* and *Exact* modes, we see that the difference is made by the type of third-party requests. For a website such as Microsoft's, these modes exhibit a similar number of requests, suggesting that third party requests are mostly for domains other than "microsoft.com". On the other hand, for websites such as Yahoo and Zoom, we see a larger amount of data transferred under *Exact* mode, suggesting that third party requests are mostly for sub-domains of the registrable part of the domain.

User Experience Analysis. During the measurement study on these 15 websites, we did not notice any broken functionality. However, more extensive testing would be required to fully assess the impact.

In addition, we measured the number of notifications made to the user by `Leakuidator`. These are measured for the interval between when the first request is made to the website and when the website finishes loading. We performed 10 independent runs for each of the 15 websites, and calculated the average number of notifications made by `Leakuidator` for each website. The results indicate that for the majority of websites the number of notifications is 0 or 1. For three websites (Yahoo, Amazon, and Microsoft), we observe 5.9 to 7.9 and 7.1 to 10 notifications in the *Relaxed* and *Exact* modes respectively. However, the majority of these notifications are for tracking requests, do not affect web functionality and user experience, and so the user can continue without making any decision.

8 Related Work

Attacks. Leaky resource attacks embed XS-leaks in benign-looking webpages. Similarly, code can be emdedded in benign-looking images, *e.g.*, SVG files [18]. Private user data may be leaked using other scriptless methods like CSS [17]. Load time measurements in an iframe HTML tag affected by service workers allow an attacker to infer whether a user visited a specific website [19].

A line of prior work targets partial deanonymization, as opposed to accurately identify a target victim. This can be achieved via browser extension fingerprinting [20,27], by inferring privacy sensitive information using shared images [13], or by leveraging browser history stealing attacks [29,32].

There are techniques for inferring whether a given user has visited a webpage controlled by an advertiser [31], but these cannot be used to track users on third-party websites. Other relevant work includes cross-site script inclusion attacks, which focuses on privacy leaks resulting from dynamically generated JavaScript [21]. Sudhodanan *et al.* [30] generalize and classify cross origin state inference attacks. We continue this line of work by improving effectiveness and reliability of HTML-only XS-leaks and by providing practical client-side defenses.

Server-Side Defenses. In Appendix B, we discuss the drawbacks of using the *SameSite* cookie attribute as a defense. Websites can share secret links with users, instead of using cookies. However, the secrecy of these links may be compromised, *e.g.* if insecure channels are used, through side-channels in browsers, or simply if users handle the links in an insecure way. An alternative is to share resources with users via user-specific secret links. This can result in website performance slow-down and may be challenging due to the mapping needed with content delivery networks for access control. In addition, users can only rely on the website UI for sharing and cannot directly share URLs.

A possible defense is to not respond to requests coming from origins that are not trusted, using the *origin* HTTP request header [11]. Another approach is using the Cross-Origin-Resource-Policy header to limit the websites that can include a specific resource [1]. However, these techniques are problematic as they do not allow cross-origin integration of authenticated resources.

Client-Side Defenses. Browsers can use information flow control techniques to ensure that no information about the outcome of a third-party request is leaked outside of the browser. For example, *tainted canvas* prevents pixel reads after a third-party image is painted on the *canvas* [9]. However, implementing information flow control in the browser can be challenging as there are a multitude of identified and unidentified leak methods in browsers, and may drastically affect the performance [12,14,16,22,23].

A possible partial solution is to separate user sessions in the browser, using the Multi-Account Containers add-on in Firefox, the Add Profile feature in Edge, or the Multiple People feature in Chrome. While not a bullet-proof solution, these can reduce the attack surface. Users can disable third-party cookies altogether. However, the current state of the web is not ready for that, and web functionality

will be broken, *e.g.*, advertisement. *ShareMeNot* can be used as a defense against tracking rooted at third party social widgets [24,25]. In contrast, `Leakuidator` is compatible with third-party tracking.

9 Conclusion

In this work, we study leaky resource attacks and show they are a serious privacy concern, even after one year since they were first revealed. We propose `Leakuidator`, a client-side defense that can be deployed right away without buy-in from browser vendors and website owners. We leave as future work conducting a user study to further assess the usability of the proposed defense.

Acknowledgments. This research was supported by the US National Science Foundation under Grants No. CNS 1801430 and DGE 1565478.

A JavaScript-Based Leaky Resource Attack

Script-Based Attack. The attack page can embed the JavaScript code shown in Fig. 7 in order to disclose information about the outcome of the SD-URL request [28]. The response to the SD-URL request is different depending on the user's state with respect to the target website. In one state, the user is able retrieve the image successfully, triggering the *onload* callback which informs the attacker that the intended victim has visited the attack page. In the other state, the user is unable to retrieve the image, trigerring the *onerror* callback.

```
<script>
  window.onload = function() {
    var img = document.getElementById("myPic");
    img.src = "State-Dependent-URL";
    img.onload = function() {
      httpReq("Attacker-controlled-URL", "target");
    }
    img.onerror = function() {
      httpReq("Attacker-controlled-URL", "not target");
    }
  }
</script>
<img id="myPic">
```

Fig. 7. Communication method using JavaScript.

B Drawbacks of Existing Defenses

The SameSite cookie attribute can be used to impose restrictions when cookies associated with a website (i.e., target website) can actually be sent to the target website. When set, this attribute can be assigned three values: *strict*, *lax*, and *none*. If it is set to *strict*, cookies are sent only when the target website matches the website currently shown in the browser's URL bar. If it set to *lax*, cookies will be sent when the condition for the *strict* value is met, but also when the website in the browser's URL bar matches the target website after a top-level navigation. This allows, for example, authentication cookies to be sent to an external website when using a single sign-on service. When the SameSite attribute is set to *none*, the browser will always send cookies along with requests to the target website.

Although setting this cookie attribute to *strict* or *lax* could limit the attack surface in theory, our findings (Sect. 4.4) show that many popular sharing services are still vulnerable, because the attribute is either set to *none*, or not enabled at all. A major reason for this is that the SameSite cookie attribute interferes with services provided by websites, because third party requests require authentication cookies being sent along when embedding the service in another website (*e.g.*, a *watch later* button on an embedded YouTube video, or personalized service such as favorite locations when embedding GoogleMaps).

References

1. Cross-Origin Resource Policy (CORP). https://developer.mozilla.org/en-US/docs/Web/HTTP/Cross-Origin_Resource_Policy_(CORP)
2. Dropbox. https://www.dropbox.com/
3. The fbi booby-trapped a video to catch a suspected tor sextortionist. https://www.vice.com/en_us/article/gyyxb3/the-fbi-booby-trapped-a-video-to-catch-a-suspected-tor-sextortionist
4. Giorgio maone. noscript. https://noscript.net/
5. Google Drive. https://www.google.com/drive/
6. Microsoft One Drive. https://www.microsoft.com/en-us/microsoft-365/onedrive/online-cloud-storage
7. Network investigative technique. https://en.wikipedia.org/wiki/Network_Investigative_Technique
8. Puppeteer. https://github.com/puppeteer/puppeteer
9. Security and tainted canvases. https://developer.mozilla.org/en-US/docs/Web/HTML/CORS_enabled_image#security_and_tainted_canvases
10. The u.s. government has withdrawn its request ordering twitter to identify a trump critic. https://www.washingtonpost.com/news/the-switch/wp/2017/04/07/the-u-s-government-has-withdrawn-its-request-ordering-twitter-to-identify-a-trump-critic
11. Verifying origin with standard headers. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#verifying-origin-with-standard-headers
12. Bauer, L., Cai, S., Jia, L., Passaro, T., Stroucken, M., Tian, Y.: Run-time monitoring and formal analysis of information flows in chromium. In: 22nd Annual Network and Distributed System Security Symposium (NDSS). The Internet Society (2015)

13. Cheung, M., She, J.: Evaluating the privacy risk of user-shared images. ACM Trans. Multimedia Comput. Commun. Appl. **12**(4s) (Sep 2016)
14. Chudnov, A., Naumann, D.A.: Information flow monitor inlining. In: 23rd IEEE Computer Security Foundations Symposium, pp. 200–214. IEEE (2010)
15. Englehardt, S., Narayanan, A.: Online tracking: a 1-million-site measurement and analysis. In: Proceedings of ACM CCS 2016, CCS 2016, pp. 1388–1401. ACM (2016)
16. Groef, W.D., Devriese, D., Nikiforakis, N., Piessens, F.: Flowfox: a web browser with flexible and precise information flow control. In: Proceedings of the ACM Conference on Computer and Communications Security, pp. 748–759. ACM (2012)
17. Heiderich, M., Niemietz, M., Schuster, F., Holz, T., Schwenk, J.: Scriptless attacks: stealing more pie without touching the sill. J. Comput. Secur. **22**(4), July 2014
18. Heiderich, M., Frosch, T., Jensen, M., Holz, T.: Crouching tiger-hidden payload: security risks of scalable vectors graphics. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 239–250. ACM (2011)
19. 0 Karami, S., Ilia, P., Polakis, J.: Awakening the web's sleeper agents: misusing service workers for privacy leakage. In: Proceedings of NDSS 221 (2021)
20. Karami, S., Ilia, P., Solomos, K., Polakis, J.: Carnus: exploring the privacy threats of browser extension fingerprinting. In: Proceedings of NDSS 2020 (2020)
21. Lekies, S., Stock, B., Wentzel, M., Johns, M.: The unexpected dangers of dynamic Javascript. In: Proceedings of the 24th USENIX Security Symposium, pp. 723–735 (2015)
22. Magazinius, J., Russo, A., Sabelfeld, A.: On-the-fly inlining of dynamic security monitors. In: IFIP International Information Security Conference, pp. 173–186 (2010)
23. Rajani, V., Bichhawat, A., Garg, D., Hammer, C.: Information flow control for event handling and the dom in web browsers. In: 2015 IEEE 28th Computer Security Foundations Symposium, pp. 366–379. IEEE (2015)
24. Roesner, F., Kohno, T., Wetherall, D.: Detecting and defending against third-party tracking on the web. In: Proceedings of USENIX NSDI 2012, pp. 155–168 (2012)
25. Roesner, F., Rovillos, C., Kohno, T., Wetherall, D.: Sharemenot: balancing privacy and functionality of third-party social widgets. In: Usenix; login (2012)
26. Schwarz, M., Lipp, M., Gruss, D.: Javascript zero: real Javascript and zero side-channel attacks. In: Proceedings of NDSS 2018 (2018)
27. Sjösten, A., Acker, S.V., Sabelfeld, A.: Discovering browser extensions via web accessible resources. In: Proceedings of the ACM CODASPY 2017, pp. 329–336 (2017)
28. Staicu, C.A., Pradel, M.: Leaky images: targeted privacy attacks in the web. In: Proceedings of the 28th USENIX Security Symposium, pp. 923–939 (2019)
29. Su, J., Shukla, A., Goel, S., Narayanan, A.: De-anonymizing web browsing data with social networks. In: Proceedings of the 26th International Conference on World Wide Web (2017)
30. Sudhodanan, A., Khodayari, S., Caballero, J.: Cross-origin state inference (COSI) attacks: leaking web site states through XS-Leaks. In: Proceedings of NDSS 2020 (2020)
31. Venkatadri, G., et al.: Privacy risks with Facebook's PII-based targeting: auditing a data broker's advertising interface. In: Proceedings of IEEE S&P 2018, pp. 89–107. IEEE (2018)
32. Wondracek, G., Holz, T., Kirda, E., Kruegel, C.: A practical attack to de-anonymize social network users. In: Proceedings of IEEE S&P 2010, pp. 223–238 (2010)