Cross-Origin State Inference (COSI) Attacks: Leaking Web Site States through XS-Leaks

Avinash Sudhodanan
IMDEA Software Institute
avinash.sudhodanan@imdea.org

Soheil Khodayari
CISPA Helmholtz Center for Information Security
soheil.khodayari@cispa.saarland

Juan Caballero
IMDEA Software Institute
juan.caballero@imdea.org

***Abstract*—In a Cross-Origin State Inference (COSI) attack, an attacker convinces a victim into visiting an attack web page, which leverages the cross-origin interaction features of the victim's web browser to infer the victim's state at a target web site. Multiple instances of COSI attacks have been found in the past under different names such as login detection or access detection attacks. But, those attacks only consider two states (e.g., logged in or not) and focus on a specific browser leak method (or XS-Leak).**

This work shows that mounting more complex COSI attacks such as deanonymizing the owner of an account, determining if the victim owns sensitive content, and determining the victim's account type often requires considering more than two states. Furthermore, robust attacks require supporting a variety of browsers since the victim's browser cannot be predicted apriori. To address these issues, we present a novel approach to identify and build complex COSI attacks that differentiate more than two states and support multiple browsers by combining multiple attack vectors, possibly using different XS-Leaks. To enable our approach, we introduce the concept of a COSI attack class. We propose two novel techniques to generalize existing COSI attack instances into COSI attack classes and to discover new COSI attack classes. We systematically apply our techniques to existing attacks, identifying 40 COSI attack classes. As part of this process, we discover a novel XS-Leak based on *window.postMessage*. We implement our approach into Basta-COSI, a tool to find COSI attacks in a target web site. We apply Basta-COSI to test four stand-alone web applications and 58 popular web sites, finding COSI attacks against each of them.

I. Introduction

In a Cross-Origin State Inference (COSI) attack, the attacker's goal is to determine the state of a *victim* visiting an *attack page* (e.g., attack.com/index.html), in a *target web site* not controlled by the attacker (e.g., linkedin.com). The state of the victim in a target web site is defined, among others, by login status, account, and content properties. Determining the victim's state can have important security implications. For example, determining that a victim is logged into a target web site implies that the victim owns an account in that site. This is problematic for privacy-sensitive web sites such as those related to post-marital affairs and pornography. Determining content ownership can be used to establish if a program committee member is reviewing a specific paper in a conference management system, or if the victim has uploaded some copyrighted content to an anonymous file sharing site. Determining if the victim owns a specific account, i.e., deanonymizing the account owner, enables identifying which company employee runs an anonymous blog criticizing the company's management. Such state inferences are even more critical when the attacker is a nation state that performs censorship and can determine if the victim has an account in, or is the administrator of, some prohibited web site. The problem is aggravated by COSI attacks being web attacks, which can be performed even when the victim employs anonymization tools such as a virtual private network.

In a COSI attack, the attacker convinces the victim to visit an attack page. The attack page includes at least one *state-dependent URL* (SD-URL) from the target web site, whose response depends on the state of the visitor. For example, a SD-URL may point to some content in the target web site only accessible when the victim has a specific state such as being authenticated. The inclusion forces the victim's browser to send a cross-origin request to the target web site. Since the request is cross-origin, the same-origin policy (SOP) prevents the attack page from directly reading the response. However, the attacker can leverage a *browser leak method* (also known as *XS-Leak*) to infer, from the cross-origin response, the victim's state at the target web site.

Multiple instances of COSI attacks have been found in the last 13 years by both security analysts (e.g., [26], [27], [33], [36], [40], [51]) and academics (e.g., [21], [31], [38], [56], [64]), with roughly half of them being presented in the last four years, and several in 2019 (e.g., [55], [56], [61]). However, they have previously been considered as sparse attacks under different names such as login detection attacks [34], [35], [51], [56], login oracle attacks [50], [57], cross-site search attacks [31], URL status identification attacks [47], and cross-site frame leakage attacks [55]. As far as we know, we are the first to systematically study these attacks and group them under the same COSI attack denomination.

Previous works have several limitations. First, they consider two states. For example, login detection attacks differentiate if the victim is logged in or not, and access detection attacks if the victim has previously accessed a site or not. However, sites typically have more than two states. Considering only two states limits the type of attacks that can be launched, and can introduce false positives, e.g., determining that a victim is logged in when he is not. A second limitation is that they often test attacks only on one browser, thus the attack may not work on other browsers. To address both issues, we present a

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24278
www.ndss-symposium.org

novel approach to identify and build complex COSI attacks by combining multiple attack vectors in order to handle more than two states and multiple browsers. For example, our approach identifies a COSI attack against HotCRP that determines if the victim, i.e., a program committee member using Chrome, Firefox, or Edge is the reviewer of a submitted paper. This attack involves multiple states (e.g., author, reviewer, logged out) and requires two COSI attack vectors: one to determine if the victim is logged in and another to determine if a logged victim is reviewing the paper.

A third limitation is that they focus on a specific XS-Leak. Instead, our approach is generic; it supports all known XS-Leaks and can easily accommodate new ones. For example, it incorporates a novel XS-Leak we have discovered based on *window.postMessage*, which affects popular sites such as blogger.com, ebay.com, reddit.com, and youtube.com. At the core of our generic approach is the concept of a *COSI attack class*, which defines the SD-URLs that can be attacked using a specific XS-Leak, the affected browsers, and the set of *inclusion methods* (i.e., HTML tags and DOM methods) that can be used to include the SD-URL in the attack page. To identify attack classes we propose a novel generalization technique that given a previously known COSI attack, generalizes it into an attack class that covers many other attack variants. We also propose an amplification technique that identifies previously unknown variations, e.g., attack classes using different inclusion methods. We systematically explore the literature to identify previously known COSI attack instances and apply our generalization and amplification techniques on them. This process identifies 40 COSI attack classes, of which 19 generalize prior attacks and 21 are new variations.

We implement our approach into a tool called Basta-COSI, publicly available as part of the open-source ElasTest platform [4]. Given as input a target web site and state scripts defining the user states at the target web site, Basta-COSI identifies SD-URLs in the target web site, tests if those SD-URLs can be attacked using any of the 40 attack classes, and produces attack pages that combine multiple attack vectors to uniquely identify a state. We have applied Basta-COSI to 62 targets: four stand-alone web applications (HotCRP, GitLab, GitHub, and OpenCart) and 58 popular web sites. Basta-COSI discovers at least one COSI attack against all of them; it finds login detection attacks against all 62 targets, account deanonymization attacks in 36, account type detection attacks in 5, SSO status attacks in 12, and access detection attacks in 5. The attacks include, among others, deanonymization attacks for determining if the victim is the reviewer of a paper in HotCRP, owns a blog in blogger.com, an account in pornhub.com, or a GitLab/GitHub repository.

The following are the main contributions of this paper:

- We present a novel approach to identify and build complex COSI attacks that differentiate more than two states and support multiple browsers. To enable our approach we propose COSI attack classes, which define the SD-URLs and browsers that can be attacked using an XS-Leak and a set of inclusion methods.
- We discover a novel XS-Leak based on *window.postMessage* that affects the three major browsers and can be leveraged to attack popular web sites.
- We propose two techniques to generalize known COSI attack instances into COSI attack classes and to discover new variations. We perform the first systematic study of COSI attacks and apply our techniques to them, identifying 40 attack classes, of which 19 generalize prior attacks and 21 are new variations.
- We implement our approach into Basta-COSI, a tool to find COSI attacks in a target web site. We apply Basta-COSI to 62 targets including stand-alone web applications and popular live sites. We find COSI attacks against all of them, enabling account deanonymization, account type inference, SSO status, login detection, and access detection.
- We have released Basta-COSI as part of the security service of the ElasTest open-source platform for testing cloud applications [4].

State Attribute	Possible Values
Login Status	(a) Logged in (b) Not logged in
Single Sign-On Status	(a) Logs in via a specific SSO service (b) Logs in via another SSO service
Access Status	(a) Has previously accessed (b) Has not previously accessed
Account Type	(a) Has a premium account (b) Has a regular account
Account Age Category	(a) Age above a certain threshold (b) Age below a certain threshold
Account Ownership	(a) Owner of a specific account (b) Not the owner of an account
Content Ownership	(a) Owner of a specific content (b) Not the owner of a content

TABLE I: Examples of user states in a target web site.

II. Overview

This Section provides an overview of COSI attacks. Section II-A details the user state at a target web site. Section II-B describes the two phases of a COSI attack. Section II-C discusses handling more than two states. Finally, Section II-D presents the COSI attack threat model.

A. User State

Most web sites have accounts owned by a user and identified by a username. In this paper a user is a person who visits a target web site and may or may not own an account in that site; it should not be confused with a username that identifies an account. Accounts are often anonymous, i.e., the person that owns the account is unknown. Deanonymizing an account means linking its username to the person owning the account. Web sites that do not have accounts often define sessions to identify users that visit them repeatedly. In those sites a session acts as an account for our purposes.

In a COSI attack, the attacker's goal is to infer the state of a victim user with respect to a target web site, not controlled by the attacker. The state of a user at a target web site is defined by the values of status, account, and ownership state attributes. Example state attributes are provided in Table I. The values of those state attributes define, at a given time, what content the user can access (or receives) from the target site. Status attributes include whether the user is logged in, logged out, logged in using a specific single sign-on (SSO) service, or has an ongoing session (i.e., in sites without user accounts). Account attributes include the account type (e.g., regular, premium, administrator), the account age category (e.g., underage user with restricted access). Ownership attributes include whether the user is the owner of some specific account and whether he owns some content stored in the site (e.g., a PDF paper in a conference management system).

The attributes that define the user's state are specific to each target site. Any of those attributes may be targeted by an attacker with different, often critical, security implications. For example, COSI attacks targeting the login status can be used by an oppressive regime to determine if the victim is logged in (and thus owns an account) in a censored site [22], despite the victim using a VPN. They can also be used to blackmail users owning accounts in privacy-sensitive sites such as those related to pornography [24] and post-marital affairs [39]. Furthermore, they may be used as an initial step for Cross-Site Request Forgery (CSRF) [20] or Cross-Site Scripting (XSS) [49] attacks. Attacks on access status have similar implications than those on login status for sites without user accounts. For example, they could be used to determine if a user previously visited a forbidden site [56].

COSI attacks targeting ownership are highly impactful. Content ownership can be used to determine if a program committee member is reviewing a specific paper, or if a user has uploaded some copyrighted content to an anonymous file sharing site. Account ownership can be used for deanonymizing the account in a closed-world setting, i.e., determining which of n known persons owns a specific account. Such closed-world deanonymization can be used to determine which company employee is the owner of an anonymous blog highly critical with the company's management.

Attacks that target account type, account age category, and login status can be used to fingerprint the victim [41], [71], and applied for targeted advertising by a malicious publisher in an open-world setting (where the set of users is unknown). Finally, knowledge of the SSO service used by the victim can be used to exploit a vulnerability in that SSO [16], [17], [66].

State scripts. In this work, we capture states at a target site using state scripts that can be executed to automatically log into the target site using a configurable browser and the credentials of an account with a specific configuration. For example, we may create multiple user accounts with different configurations, e.g., premium and free accounts, two users that own different blogs, or authors that have submitted different papers to a conference management system. We also create a state script for the logged out state.

B. COSI Attack Overview

In a COSI attack, the attacker convinces a victim to visit an attack page. The attack page leverages the cross-origin functionalities of the victim's web browser to infer the victim's state at a target web site. A COSI attack comprises of two phases: *preparation* and *attack*.

Preparation. The goal of the preparation phase is to create an attack page that when visited by a victim will leak the victim's state at the target web site. An attack page implements at least one, possibly more, attack vectors. Each attack vector is a triplet of a *state-dependent URL* from the target web site, an *inclusion method* to embed the SD-URL in the attack page, and an *attack class* that defines, among others, a *leak method* (or *XS-Leak*) that interacts with the victim's browser to disclose a victim's state at the target site. An attack page may contain multiple attack vectors. For example, it may need to chain attack vectors to uniquely distinguish a state, e.g., one to identify if the victim is logged in, and another to identify if a logged victim has a premium account.

We say that a URL is state-dependent if, when requested through HTTP(S), it returns different responses depending on the state it is visited from. Note that it is not needed that each state returns a different response. For example, if there are 6 states and two different responses, each for three states, the URL is still state-dependent. The SD-URL is included by the attack page using an inclusion method such as an HTML tag (e.g., *img*, *script*) or a browser DOM method (e.g., *window.open*). When the attack page is visited by the victim, the inclusion method forces the victim's browser to automatically request the SD-URL from the target site. The specific response received depends on the victim's current state. SD-URLs are very common in web applications. For example, in many web applications, sending a request for a profile's picture will return an image if the user is logged in, and an error page, or a redirection to the login page, otherwise. Similarly, in a blog application, a new post can only be added if the user is both logged in and the owner of the blog.

The request induced by the attack page for a SD-URL at the target site is cross-origin, and thus controlled by the Same-Origin Policy (SOP) [72]. The SOP prevents the attack page from directly reading the contents of a cross-origin response [18]. However, there exist XS-Leaks that allow bypassing a browser's SOP to disclose information about cross-origin responses. For example, the *EventsFired* XS-Leak distinguishes responses to SD-URLs that trigger a callback in one state (e.g., *onload*) and another callback (e.g., *onerror*), or no callback, in another state [36].

While a target site may contain many SD-URLs, only a subset of those may be useful to mount a COSI attack. One main challenge with XS-Leaks is that their behavior may depend on the target browser and the inclusion method used. Unfortunately, this key concept is missing from prior works presenting COSI attack instances. In this work, we introduce the concept of a *COSI attack class*, which defines the two different responses to a SD-URL that can be distinguished using a XS-Leak, the possible inclusion methods that can be used in conjunction with the XS-Leak, and the browsers affected. Attacks classes are independent of the target site states and thus can be used to mount attacks against different targets. Section III describes our approach to identify attack classes and the 40 COSI attack classes we identified.

Based on the attack classes, we propose a novel approach to detect COSI attacks. Our approach first collects the responses to the same URL from different states. SD-URLs will be the ones that produce different responses in some states. Each pair of different responses coming from distinct states is matched with the list of known attack classes. If a matching attack class is found, then an attack vector can be built to distinguish the responses (and thus the states that produce them) that uses that SD-URL, the XS-Leak in the attack class, and one of the inclusion methods defined by the attack class. Since there may be $n > 2$ states that need to be distinguished, the process repeats until sufficient attack vectors are identified to uniquely distinguish the target state to be attacked. We have implemented this approach into Basta-COSI, a tool to detect COSI attacks, detailed in Section IV.

Attack. In the attack phase, the attacker convinces the victim into visiting the attack page. This can be achieved in multiple ways. One possibility is sending an email with the attack page URL and text to convince the victim to click on it. Such targeted attack requires the victim's email, but allows identifying the state of a specific person, e.g., deanonymizing the owner of an account. Another possibility is a watering-hole approach where the attacker injects the attack page URL into a vulnerable page that victims are likely to visit. Such attack allows identifying the state of a visitor, but does not identify who the visitor is. The method used to convince the victim to visit the attack page is outside the scope of this paper. When the attack page is loaded at the victim's browser, it checks the browser used by the victim, delivers suitable attack vectors, and reports back the leaked victim's state.

C. Beyond Two States

Current COSI attacks targeting login or access detection consider only two states. However, most web sites have more than two states, e.g., logged in users with different permissions. Considering only two states introduces some issues. First, it limits the type of attacks, preventing attacks that target finer-grained states such as account type or content ownership. Furthermore, it can introduce false positives, which is best illustrated with an example.

In 2015, Lee et al. [47] presented a novel AppCache XS-Leak (described in Section III) that enabled login detection. One of their login detection attacks targeted the NDSS 2015 HotCRP installation. The SD-URL https://ndss2015.ccs.neu.edu/paper/⟨paper-no⟩ returned a success HTTP status code when the victim was logged into HotCRP and an error status code otherwise. That difference could be identified using the AppCache XS-Leak. In reality, the HotCRP access control is more fine-grained and the information of a paper can only be accessed by its authors or by reviewers, but not by other authors who would also receive an error. Thus, their attack could incorrectly identify an authenticated victim, who happened to be an author of another paper, as not being authenticated. Such false positives could be avoided if they could guarantee that victims would not be authors (e.g., not sending authors an email with the attack page URL), but authors are only known to the conference administrators.

Running example. As running example we use a reviewer deanonymization attack Basta-COSI found on HotCRP, which was acknowledged and fixed. Listing 1 shows a simplified version of the attack page produced by Basta-COSI that we sent to HotCRP developers to report the attack. It identifies if the visiting victim is the reviewer of paper #123 submitted to https://conf.hotcrp.com. Since HotCRP has multiple states (e.g., logged in, author, reviewer, reviewer of a specific paper) and we want to support the major browsers (Chrome, Firefox, Edge), the attack page requires three attack vectors executed when the attack page is loaded (Line 3). It first runs an

Listing 1: Running example attack page for deanonymizing the reviewer of a paper in HotCRP.

```
1  <!DOCTYPE html><html>
2  //Launch attack when page loads
3  <body onload="attack()"><script>
4  //SD-URLs used in the attack vectors
5  site = "https://conf.hotcrp.com"
6  loginURL = site+"/offline.php?downloadForm=123";
7  reviewURL = site+"/api.php/review?p=123";
8  //Object for storing fired events
9  evnts = {"obj": [], "lnk" : [], "embd" : []}
10 function attack() {
11   // Login detection on all browsers
12   EF_XctoObject();
13   // Reviewer deanonymization
14   if (detectBrowser() == "Chrome") {
15     EF_StatusErrorLink();
16   }
17   else { EF_StatusErrorObject(); }
18   sendToAttkr(evnts); //send events to attacker
19 }
20 function EF_XctoObject() {
21   tag = document.createElement("object");
22   tag.setAttribute("data", loginURL);
23   tag.setAttribute("rel", "stylesheet");
24   tag.onload = function(){
25     evnts["obj"].push("onload");
26   }
27   document.body.appendChild(tag);
28 }
29 function EF_StatusErrorLink(){...}
30 function EF_StatusErrorObject(){...}
31 </script></body></html>
```

attack vector for determining the victim's login status, which works regardless if the victim's browser is Chrome, Firefox, or Edge (Lines 12, 20-28). This attack vector includes SD-URL https://conf.hotcrp.com/offline.php?downloadForm=123 with the `object` HTML tag and uses the *EventsFired* XS-Leak: if the victim is logged into the site, no events are triggered, otherwise the *onload* event is triggered. Then, it executes the attack vectors for reviewer deanonymination, which differ for Chrome (Line 15) and Firefox/Edge (Line 17). These attack vectors are not detailed for brevity, but both use the *EventsFired* XS-Leak with different inclusion methods for the same SD-URL https://conf.hotcrp.com/api.php/review?p=123, which returns a success HTTP status code if the victim has submitted a review for paper #123, and an error HTTP status code otherwise.

D. Threat Model

This section describes the COSI attack threat model, detailing the assumptions we make about each actor.

Attacker. We assume that the attacker can trick victims into loading the attack page on their web browsers. During preparation, the attacker has the ability to create and manage different accounts at the target web site, or in a local installation of the target's web application. The attacker controls an attack web site where he can add arbitrary pages. Finally, we assume the attacker can identify the victim's browser version (e.g., from the User-Agent header) to select the right attack vector.

Victim. The victim uses a fully up-to-date web browser and can be lured by the attacker into visiting the attack webpage.

We assume that the victim logs into the target web site with the same web browser used to visit the attack page.

Target site. The target site contains at least one SD-URL for which the attacker knows an attack class. The target site does not suffer from any known vulnerabilities. In particular, resources containing sensitive information are protected from direct cross-origin reads, i.e., the target site does not contain CORS misconfigurations [48], cross-site scripting [49], or cross-site script inclusion vulnerabilities [50].

III. COSI Attack Classes

A key concept in our approach are COSI attack classes. A COSI attack class is a 6-tuple that comprises of a class name, signatures for two groups of responses that can be distinguished using the attack class, an XS-Leak, a list of inclusion methods that can be used to embed the SD-URL in an attack page, and the list of affected browsers. It captures which SD-URLs can be used for building an attack vector against the affected browsers using the XS-Leak and one of the inclusion methods defined. A reader could think that an attack class should simply correspond to an XS-Leak. However, the behavior of some XS-Leaks depends on the target browser and the inclusion method used. Depending on those two parameters, the set of affected SD-URLs differs. Thus, identifying attack classes is fundamental for determining whether and how a given SD-URL can be attacked. This section first presents our approach to discover COSI attack classes in Section III-A and then details the 40 attack classes identified in Section III-B.

A. Discovering Attack Classes

Our process to discover COSI attack classes comprises of three main steps: (1) identify and validate previously proposed COSI attack instances; (2) generalize known COSI attack instances into COSI attack classes; and (3) discover previously unknown attack classes.

Identifying attack instances. We have performed a systematic survey of COSI attack instances presented in prior work under different names. This process identified 23 prior works, listed in Table VIII and described in Section IX. Out of those, 11 are blog posts, 10 are academic papers, one is a bug report, and the last one is a project simultaneous to our work that tries to enumerate all known XS-Leaks [65]. Those 23 prior works presented 31 attack instances. All attack instances could be validated in at least one recent browser version. To validate an attack instance we manually create a test attack page based on the available information. The test attack page includes a URL from a test application we have designed to return custom responses to an incoming request. Requests to the test application define how the response should look (i.e., which headers and body to return). In this step, we configured our test application to return the responses described in the work presenting the attack. This enables validating attack instances even when the SD-URL used in the attack was no longer active.

Generalizing instances into classes. Generalizing a COSI attack instance into a COSI attack class comprises of two steps. First, identifying the set of responses to the inclusion method used in the attack instance, that still trigger the same observable difference in the browser (e.g., onload/onerror or different object property values). Then, checking if the observable difference still manifests with other inclusion methods and browsers. The generalization uses the test application to control the response received from a potential target site. We illustrate it using an attack instance of the *EF-StatusErrorObject* attack class. The generalization starts with the response that triggers the onload callback and tries to modify each response element (header or body) to a different value. If the modification still triggers the onload callback, then the element can be ignored. In our example, all fields can be ignored, except the status code that it should be 200 and the content-type that should not correspond to an audio or video. The generalization then repeats for the response that triggers the onerror callback, returning that the status code should not be success (200) or redirection (3xx), but other values for the status code, headers, and body do not matter. Once the responses are generalized, it tests whether other inclusions methods still trigger the same observable difference. For this, it tests the *window.open()* method and the 13 HTML tags that enable resource inclusion without user intervention, shown in Table II. Finally, it checks if the leak manifests in other browsers. Table VIII shows that the 31 attack instances examined belonged to 15 attack classes, i.e., many were duplicates.

Tag	Attribute	Included Resource's Type
`applet`	`code`	Applet
`audio`	`src`	Audio
`embed`	`src`	Defined in `type` attribute
`frame`	`src`	Typically web pages
`iframe`	`src`	Typically web pages
`img`	`src`	Image
`input`	`src`	Image (when attr. `type` = "picture")
`link`	`href`	Defined in `rel` and `type` attributes
`object`	`data`	Defined in `type` attribute
`script`	`src`	JS
`source`	`src`	Audio/Video
`track`	`src`	WebVTT [8]
`video`	`poster`	Image
`video`	`src`	Video

TABLE II: HTML tags supporting resource inclusion.

Discovering new attack classes. The test application allows systematically exploring combinations of header and body values in responses. For each response, browser events and DOM values are logged. Pairs of responses that produce observable differences (e.g., trigger different callbacks), and do not match existing attack classes, correspond to new attack instances, and are generalized as above. Overall, we discovered 21 new attack classes, of which 12 use the EventsFired (i.e., onload/onerror) XS-Leak, 8 use the Object Property XS-Leak, and 1 uses a completely novel XS-Leak based on postMessage.

B. Attack Classes Description

Table III details the 40 attack classes identified by the above process. For each attack class, the table shows the name we assigned to the class; a description of the two different responses by a SD-URL that can be targeted using this attack class; the attack page logic with the methods that can be used to include the SD-URL and the XS-Leak to distinguish the responses; and the affected browsers. In each response description we abbreviate HTTP fields as follows:

Class	SD-URL Responses		Attack Page's Logic		Browsers		
	Response A	*Response B*	*Inclusion Methods*	*Leak Method*	*Firefox*	*Chrome*	*Edge*
EF-StatusErrorScript	sc = 200, ct = text/javascript	sc = (4xx OR 5xx)	`script src=URL`	[onload] / [onerror]	✓	✓	✓
EF-StatusErrorObject	sc = 200, ct ≠ (audio OR video)	sc ≠ (200 OR 3xx)	`object data=URL`	[onload] / [onerror]	✓	✗	✗
EF-StatusErrorEmbed	sc = 401, ct = (text/html)	sc ≠ 401, ct = (text/html)	`embed src=URL`	[] / [onload]	✗	✗	✓
EF-StatusErrorLink	sc = (200 OR 3xx), ct ≠ text/html	sc ≠ (200 OR 3xx)	`link href=URL rel=prefetch`	[onload] / [onerror]	✗	✓	✗
EF-StatusErrorLinkCss	sc = (200 OR 3xx), ct = text/css	sc ≠ (200 OR 3xx), ct ≠ text/css	`link href=URL rel=stylesheet`	[onload] / [onerror]	✓	✓	✗
EF-RedirStatLink	sc = 3xx	sc ≠ 3xx, cto = nosniff, ct ≠ (text/css OR text/html)	`link href=URL rel=stylesheet`	[onload] / [onerror]	✗	✓	✗
EF-StatusErrorIFrame	sc = (200 OR 3xx OR 4xx or 5xx), ct= (text/javascript OR text/css)	sc = (200 OR 3xx OR 4xx or 5xx), ct ≠ (text/javascript OR text/css)	`iframe src=URL`	[] / [onload]	✗	✗	✓
EF-NonStdStatusErrorIFrame	sc = (200 OR 3xx OR 4xx or 5xx), ct = (text/javascript OR text/css)	sc = 999	`iframe src=URL`	[] / [onload]	✗	✗	✓
EF-CDispIFrame	sc = 200, cd = attachment	cd ≠ attachment	`iframe src=URL`	[] / [onload]	✗	✓	✗
EF-CDispStatErrIFrame	sc = (4xx OR 5xx), cd = attachment	sc = (4xx OR 5xx), cd ≠ attachment	`iframe src=URL`	[] / [onload]	✓	✗	✗
EF-CDispAthmntIFrame	sc = 200, cd = attachment	¬(sc = 200, cd = attachment)	`iframe src=URL`	[] / [onload]	✗	✓	✗
EF-XctoScript	sc = 200, xcto disabled, ct = (text/html OR text/css OR application/pdf)	sc = 200, xcto = nosniff, ct = (text/html OR text/css OR application/pdf)	`script src=URL`	[onload] / [onerror]	✓	✗	✓
EF-XctoObject	sc = 200, xcto disabled, ct = (text/html OR text/css OR application/json)	sc = 200, xcto = nosniff, ct = (text/html OR text/css OR application/json)	`object data=URL`	[onload] / [ ]	✓	✓	✓
EF-CtMismatchObject	sc = 200, ct = X	sc = 200, ct = Y	`object data=URL typesmustmatch type=X`	[onload] / [onerror]	✓	✗	✗
EF-CtMismatchScript	sc = 200, ct = (text/javascript)	sc = 200, xcto = nosniff, ct ≠ (text/-javascript)	`script src=URL`	[onload] / [onerror]	✓	✗	✓
EF-CtMismatchImg	sc = (200 OR 3xx OR 4xx OR 5xx), ct = image	sc = (200 OR 3xx OR 4xx OR 5xx), ct ≠ image	`img src=URL`	[onload] / [onerror]	✗	✓	✓
EF-CtMismatchAudio	sc = (200 OR 3xx OR 4xx OR 5xx), ct = audio	sc = (200 OR 3xx OR 4xx OR 5xx), ct ≠ audio	`audio src=URL`	¬[onerror OR on-suspend] / [onerror OR onsuspend]	✗	✓	✗
EF-CtMismatchVideo	sc = (200 OR 3xx OR 4xx OR 5xx), ct = video	sc = (200 OR 3xx OR 4xx OR 5xx), ct ≠ video	`video src=URL`	¬[onerror OR on-suspend] / [onerror OR onsuspend]	✓	✗	✗
EF-XfoObject	sc = 200, xcto = text/*, xfo is disabled	sc = 200, xfo is enabled	`object data=URL`	[] / [onload]	✗	✓	✗
EF-CacheLoadCheck	bdy = includes URL A	bdy = does not include URL A	`Send error req to URL A, link rel=preload href=URL, img src=URL A, send error req to URL A`	[onload]/[onerror]	✓	✓	✗
OP-LinkSheet	sc = 200, ct = text/css, bdy = CSS-like	sc = 200, ct ≠ text/css, bdy ≠ CSS-like	`link rel=stylesheet href=URL`	sheet	✗	✗	✓
OP-LinkSheetStatusError	sc = (200 OR 3xx), ct ≠ text/css	sc ≠ (200 OR 3xx)	`link rel=stylesheet href=URL`	sheet	✗	✗	✓
OP-ImgDimension	sc = (200 OR 3xx OR 4xx OR 5xx), ct = image, bdy = image with dimension *A*	sc = (200 OR 3xx OR 4xx OR 5xx), ct = image, bdy = image with dimension *B*	`img src=URL`	height, width, naturalHeight, naturalWidth	✓	✓	✓
OP-VideoDimension	sc = (200 OR 3xx OR 4xx OR 5xx), bdy = video with dimension *A*	sc = (200 OR 3xx OR 4xx OR 5xx), body = (video with dimension *B* OR body not video)	`video src=URL`	videoHeight, videoWidth	✓	✓	✓
OP-WindowDimension	sc = (200 OR 3xx OR 4xx OR 5xx), bdy = PDF	sc = (200 OR 3xx OR 4xx OR 5xx), body ≠ PDF	`frame src=URL`	height, width	✗	✗	✓
OP-MediaDuration	sc = 200, ct = (audio or video), bdy = audio/video with duration *A*	sc = 200, ct = (audio OR video), bdy = audio/video with duration *B*	`audio/video src=URL`	duration	✓	✓	✓
OP-ImgCtMismatch	sc = 2xx, ct = image	sc = 4xx, ct ≠ image	`img src=URL`	height, width, naturalHeight, naturalWidth	✓	✗	✓
OP-MediaCtMismatch	sc = 200, ct = (audio OR video)	ct ≠ (audio OR video)	`audio/video src=URL`	networkState, readyState, buffered, paused, duration, seekable	✓	✓	✓
OP-FrameCount	sc = 200, ct = text/html, bdy = HTML with numFrames *A*	sc = 200, ct = text/html, xfo is disabled, bdy = HTML with numFrames *B*	`iframe src=URL`, (`form`, `iframe`)	contentWindow.length	✓	✓	✓
OP-MediaStatus	sc = 2xx, ct = (audio OR video)	sc = 4xx OR 5xx ct ≠ (audio OR video)	`video/audio src=URL`	error.message	✓	✗	✗
OP-XfoObject	sc = 200, xfo is disabled, ct = text/*	sc = 200, xfo is enabled	`object data=URL`	contentDocument	✓	✗	✗
OP-XfoIFrame	xfo is disabled	sc = (2xx OR 3xx OR 4xx OR 5xx), xfo is enabled	`iframe src=URL`	contentDocument	✓	✗	✗
OP-WindowProperties	sc = 200, ct = text/html, bdy = HTML with window property *A*	sc = 200, ct = text/html, bdy = HTML with window property *B*	`window.open(), (form, iframe)`	frames.length	✓	✓	✓
postMessage	bdy = postmsg *A* broadcast	bdy = (postmsg *B* broadcast OR no postmsgs broadcast)	`iframe`, `window.open()`	receiveMessage()	✓	✓	✓
CSSPropRead	sc = 200, ct = text/css, bdy = CSS with rule A	sc = 200, ct = text/css, bdy = CSS with rule B	`link rel=stylesheet href=URL`	window.getComputedStyle()	✓	✓	✓
JSError	sc = 200, ct = text/javascript, bdy = JS with *A* no. of errors	sc = 200, ct = text/javascript, bdy = JS with *B* no. of errors	`script src=URL`	window.onerror()	✓	✓	✓
JSObjectRead	sc = 200, ct = text/javascript, bdy = JS with readable object *A*	sc = 200, ct = text/javascript, bdy = JS with readable object *B*	`script src=URL`	window.hasOwnProperty(), prototype tampering, global API redefinition	✓	✓	✓
CSPViolation	sc = 3xx, Location = same origin	sc = 3xx, Location = different origin	`iframe, frame, embed, applet, video, audio, object, link, script`	{“csp- report”:}	✓	✓	✓
AppCacheError	sc = 200	sc = (3xx OR 4xx OR 5xx)	`html manifest=MANIFEST.appcache`	AppCache error	✗	✓	✗
Timing	Load/Resp./Parse time *A*	Load/Resp./Parse time *B*	`script, video, img, XmlHttpRequest...`	timing side-channel	✓	✓	✓

TABLE III: COSI attack classes.

Status Code (sc), Content-Type (ct), X-Content-Type-Options (xcto), Content-Disposition (cd), and response body (bdy).

EventsFired. The first 20 attack classes use the events fired in the browser as XS-Leak and hence are denoted by the prefix *EF-*. The first attack class *EF-StatusErrorScript* can target SD-URLs that return in one state a success status code ($sc = 200$) with JavaScript (JS) content ($ct = text/javascript$), and return an error ($sc = (4xx\, OR\, 5xx)$) in another state. The events fired by both types of responses are different (onload in one case, onerror in the other) allowing to distinguish the two responses. This attack class works on all browsers. Among these 20 attack classes, 14 are new and for the other 6 attack instances had been previously proposed. Most of these 20 involve the type or disposition of the content, including content-sniffing (`X-Content-Type-Options`). There are also cases related to the `X-Frame-Options` header.

Object Properties. The next 13 attack classes leverage as XS-Leak the readable properties of the included resource. Out of these 13, 8 are new variations. For instance, in *OP-ImgDimension*, if a SD-URL returns images with different dimensions, the *height* and *width* properties allow to differentiate the responses. While these two properties were known to leak [65], our approach uncovers that similar attacks exist using the *naturalHeight* and *naturalWidth* properties. Interestingly, *OP-ImgCtMismatch* presents a similar attack targeting SD-URLs that return an image and a non-image, which works because for non-image resources some browsers return the height and width of a broken image icon, triggering a difference in dimensions. The term (`form`, `iframe`) in classes *OP-FrameCount, OP-WindowProperties* captures that it is also possible to include the resource using a `form` tag (using the `action` attribute) to trigger a POST request (specifying `method` as POST), and embedding the response in an iframe (pointing `target` attribute to an iframe) [27]. All other attack classes leverage GET requests.

PostMessage. This class uses a novel XS-Leak that as far as we know has not been previously mentioned. It can target SD-URLs that return different broadcasted postMessages, or a broadcast postMessage and no broadcast. It affects all three browsers. To read the postMessages, the attack page can include the SD-URL using the *iframe* tag if the page does not use framing protection, or the *window.open* method if framing protection is used. To identify a difference between responses, it compares the number of broadcast messages, the message origins, and the message content. The message content is compared using the Jaro string distance [44] to account for small session-specific or user-specific differences.

CSSPropRead. Another XS-Leak leverages SD-URLs that return different CSS rules for different states. To identify the differences, the attack page is designed to contain elements affected by the differing rules and to check the inherited style rules. Some attack instances in this class were previously known [26], [35]. This class complements the *OP-LinkSheet* and *OP-LinkSheetStatusError* classes, which can differentiate between CSS and non-CSS responses.

JSError. When a SD-URL returns different JavaScript files, where one contains a JS error and the other does not, this difference can be detected using the *window.onerror()* callback function. The original attack instance used *window.onerror()* to read the line number and the type of JS error triggered [33]. But, since Cross-Site Script Inclusion (XSSI) attacks [32], [63] abused the verbosity of *window.onerror()*, popular browsers no longer return the error line. However, we find the attack still works by comparing the number of errors triggered. This class complements *EF-StatusErrorIFrame*, which allows differentiating JS and non-JS responses.

JSObjectRead. Another XS-Leak for differentiating responses that contain JS files checks the presence or absence of certain readable objects in the included JS. The original attack instance checked for global variables [32], but later attacks also leveraged techniques such as prototype tampering and global API redefinition [50].

CSPViolation. When a SD-URL redirects visitors to the same origin in a state and to a different origin in another state, this difference can be detected using a Content Security Policy (CSP). The attacker configures its attack site with a CSP policy for the attack page that states that any attempt to load a resource from an origin different than the attack site should send a violation report back to the attack site. This method was originally proposed for leaking sensitive information in the CSP report (e.g., in the path and subdomain) [40]. Browsers then removed the path information from CSP reports, but the attack still works by focusing on whether the CSP violation report is received (redirection to different origin) or not (redirection to same origin).

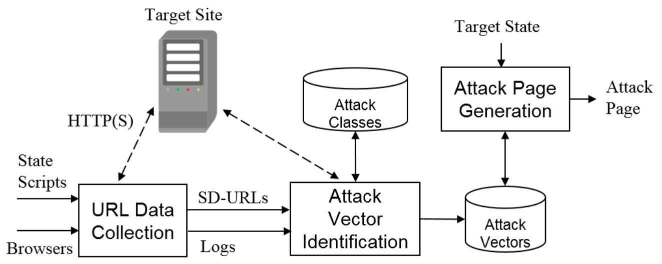
AppCacheError. When a SD-URL returns a success status code (2xx) in one state and a redirection (3xx) or error (4xx, 5xx) in another, this difference can be detected through the browser's AppCache [9]. The attack page uses the `manifest` attribute of the `html` tag to refer to an AppCache manifest file, which includes the SD-URL in the list of URLs that should be cached. This forces the browser to request the SD-URL. If the SD-URL returns a success status code, an AppCache *cached* event is triggered. If the SD-URL returns a redirection or error, an AppCache *error* event is triggered instead. Lee et al. [47] first presented this attack showing that it affected five browsers. However, this XS-Leak currently only works in Chromium-based browsers because Firefox and Edge no longer allow cross-origin URLs to be cached using AppCache.

Timing. Multiple works have shown that timing differences when a resource is requested from different states can be used to distinguish those states [21], [27], [31], [56], [64]. Those works focus on acquiring accurate timing information resistant to changes in network conditions. We have incorporated into Basta-COSI the ability to gather accurate timing information using the video parsing leak in [64].

IV. Basta-COSI

We have designed and implemented Basta-COSI, a tool for assisting a security analyst in identifying, and generating evidence of, COSI attacks in a target site. Basta-COSI focuses on the COSI attack preparation phase. It takes as input a target site, a set of state scripts defining states in the target site, and the attack classes identified in Section III. It outputs attack pages, which can be used by a security analyst for demon-

Fig. 1: Basta-COSI architecture.



strating the existence of complex COSI attacks, involving more than two states and supporting multiple browsers.

Setup. Basta-COSI needs network access to the target site, which may be a local installation of an open-source web application (e.g., GitLab, HotCRP) or a remote web site (e.g., linkedin.com, facebook.com). The analyst needs to be able to create user accounts in the target site. Those accounts should cover different account types and should be populated with content, e.g., filling the user profile, creating a blog, adding blog entries. For example, to test the open source HotCRP conference management system, the analyst prepares a local installation by creating a test conference and five user accounts: administrator, two authors, and two reviewers. Then, he submits a paper using each of the author accounts. Finally, it assigns the paper submitted by the first author to the first reviewer and the paper submitted by the second author to the second reviewer.

Once the target site is configured, the analyst creates state scripts that can be executed to automatically load a specific state at a web browser, i.e., to log into the tested web application using one of the created accounts or to log out of an account. Basta-COSI currently supports state scripts written using the Python Selenium WebDriver [6]. The web browser to be used is an argument to the state script. In our HotCRP example, the analyst creates six state scripts. The first five scripts open a web browser, visit the login page, and authenticate using one of the created accounts. The last script logs in and then logs out to capture the logged out state.

Architecture. The architecture of Basta-COSI is shown in Figure 1. It takes as input the state scripts, a set of browsers, the configured target site, and a target state. It outputs an attack page that leaks if a victim is in the target state at the target site. Basta-COSI comprises of three modules: *URL data collection*, *attack vector identification*, and *attack page generation*.

The URL data collection module crawls the target site to discover URLs. It visits each discovered URL to collect its response when visited from a specific state with a specific browser. And, it compares the responses to the same URL obtained from different states to identify SD-URLs that may be candidates to be used in attack pages.

Next, the attack vector identification checks if any of the SD-URLs can be attacked using the known COSI attack classes. When needed, it visits each SD-URL using a set of inclusion vectors to collect browser events that can only be obtained with a specific inclusion method (e.g., postMessages), or that cannot be easily obtained statically from the HTTP(S) responses (e.g., JS errors, readable JS objects). For each SD-URL that matches an attack class, it outputs an attack vector.

Finally, the attack page generation module builds an attack page that enables identifying if the victim is in the target state at the target site. The generated attack page may combine multiple attack vectors to uniquely distinguish the target state and to support multiple browsers. Attack pages for different target states can be created by re-running the attack page generation module, without re-running the previous modules.

A. URL Data Collection

The URL data collection module performs three main tasks: crawling to discover URLs, collecting the responses for each URL when visited from a specific state with a specific browser, and identifying SD-URLs. The module is built on top of the Spider crawler for OWASP ZAP [5]. The crawling considers a URL to be part of the target site if it satisfies at least one of three constraints: it is hosted at the target site domain, it redirects to a URL hosted at the target site domain, or it is part of a redirection chain involving a URL satisfying any of the above two criterion.

Each discovered URL is visited from each input state and using each input browser. Before visiting a URL, a state script is executed to load the corresponding state in the browser. The state scripts also allow collecting URLs only accessible from authenticated states. Currently, Basta-COSI supports the three most popular browsers: Chrome, Firefox, and Edge. For each browser, it supports the latest version at the time we started the implementation: Google Chrome 71.0.3578.98, Mozilla Firefox 65.0.1, and Microsoft Edge 42.17134.1.0. The module has a flexible design that allows adding support for other browsers and browser versions. For each triplet (URL, browser, state), it stores the full response (headers and body) received from the server. URLs that return the same response in each state are not state-dependent and thus cannot be used in a COSI attack. To identify if a URL is state-dependent, a similarity function is used that compares responses ignoring non-deterministic fields such as the `Date` header or CSRF tokens that may differ in each response. URLs that return the same response (minus non-deterministic fields) in every state are not state-dependent, and can be discarded.

To illustrate the tool we use our HotCRP running example with only three state scripts: Reviewer1 (R1), Reviewer2 (R2), and LoggedOut (LO). The goal of the analyst is to find a COSI attack that reveals the reviewer of a specific paper. In this scenario, the tester can ignore the administrator and author accounts since an attacker (typically an author) would only send emails with the attack page URL to the (non-chair) PC members. The three identified URLs in our running example are shown in Table IV. Each table entry shows the response for the URL when visited from a specific state. For simplicity, each response is summarized as a tuple of 4 field values: Status Code (sc), Content-Type (ct), X-Frame-Options (xfo), and X-Content-Type-Options (xcto). The URL /images/pdffx.png is not a SD-URL since it returns the same response in all states. Thus, it will be removed at this step. The other two URLs are state-dependent since for each of them there exists at least one pair of states whose responses are different.

URL	Response Received at Different States		
	Reviewer1 (R1)	*Reviewer2 (R2)*	*Logged Out (LO)*
/testconf/images/pdffx.png	sc = 200, ct = image/png, no xfo, no xcto	sc = 200, ct = image/png, no xfo, no xcto	sc = 200, ct = image/png, no xfo, no xcto
/testconf/api.php/review?p=1	sc = 200, ct = text/html, no xfo, xcto = nosniff	sc = 403, ct = text/html, no xfo, no xcto	sc = 200, ct = text/html, no xfo, no xcto
/testconf/offline.php?downloadForm=1	sc = 200, ct = text/html, no xfo, xcto = nosniff	sc = 200, ct = text/html, no xfo, xcto = nosniff	sc = 200, ct = text/html, no xfo, no xcto

TABLE IV: Examples of URLs collected from HotCRP from three states. For simplicity, the response is represented with only a subset of 4 field values: Status Code (sc), Content-Type (ct), X-Frame-Options (xfo), and X-Content-Type-Options (xcto).

B. Attack Vector Identification

The goal of the attack vector identification module is to find, among all the SD-URLs discovered, the ones for which a matching attack class is known, and thus can be used to generate attack vectors. Basta-COSI supports all attack classes in Table VIII. Those attack classes can be split into two groups. The first (static) group are attack classes for which it can be determined, using solely the collected logs of HTTP(S) responses, if a SD-URL matches the class. This group includes all classes that capture differences in HTTP headers such as Status Code, Content-Type, or X-Frame-Options. The second (dynamic) group are attack classes for which matching a SD-URL requires data difficult to obtain from the responses such as JS errors, postMessages, and audio/video properties (e.g., width, height, duration). For this group, it is needed to visit the SD-URL with different inclusion methods to collect the missing data.

For each SD-URL and pair of states that return different responses for that SD-URL, the module first checks if there exist any matching static attack classes. For efficiency, if two different state pairs produce the same responses, there is no need to query the attack classes for the second pair. We illustrate this process using the SD-URLs in Table IV. For `api.php`, the responses from (R1, R2) match two static attack classes: *EF-StatusErrorObject* (for Firefox and Edge), *EF-StatusErrorLink* (for Chrome). Similarly, the responses from (R2, LO) match the same two static attack classes as (R1, R2). Finally, the states (R1, LO) match the static attack classes *EF-XctoObject* and *EF-XctoScript*. The process repeats with the other SD-URL (`offline.php`). Since states R1 and R2 return the same response, (R1, R2) can be ignored. For states (R1, LO), the attack classes *EF-XctoObject* and *EF-XctoScript* match. Finally, for states (R2, LO) the responses are the same as for (R1, LO) and there is no need to check them again.

In our example, all state pairs can be distinguished using a static attack class. If that was not the case, the module would collect additional information to check the dynamic attack classes. For this, the SD-URL is included in a set of data collection pages hosted at a test web server. Each page uses an inclusion method from one of the dynamic classes and collects the required dynamic data for the class (e.g., use *script* to collect JS errors and JS readable objects). Each data collection page is visited with each browser and from every state that returns a unique response.

The attack vector identification module outputs, for each pair of states, a list of pairs (SD-URL, AttackClass) specifying that an attack vector that uses the SD-URL and the attack class can distinguish those two states for the browsers defined by the attack class.

Algorithm 1: Attack vector selection

```
inputs : Target state $s_t$, target browsers $B$, states $S$, attack vectors $A$
outputs: The list of selected attack vectors
1  outVectors ← [ ];
2  $S_r \leftarrow S - s_t$;
3  $A_r \leftarrow$ filter($A$, $s_t$);
4  $A_r \leftarrow$ mergeStates($A_r$);
5  $P \leftarrow (s_i \in S_r, b_j \in B)$;
6  while $P \neq \emptyset, A_r \neq \emptyset, s > 0$ do
7      V = score($A_r$, $P$);
8      (s,a) ← (max(V),argmax(V));
9      if $s > 0$ then
10         outVectors.append(a);
11         $P \leftarrow P$ - getCoveredPairs($a$);
12         $A_r \leftarrow A_r - a$;
13     end
14 end
15 return outVectors, $P$ ;
```

C. Attack Page Generation

Given a target state s_t and a set of target browsers B, the goal of the attack page generation is to produce an attack page that combines attack vectors to uniquely distinguish s_t from the other states, when visited by a browser in B. The set of target browsers should be equal to or a subset of the set of browsers input to Basta-COSI. This process comprises of two steps: attack vector selection and attack page construction.

Algorithm 1 details the attack vector selection. It selects, among all attack vectors, the ones needed to distinguish the target state when visited by a target browser. The algorithm first removes all attack vectors that do not include the target state since they do not enable distinguishing s_t (Line 3). In our HotCRP example, the target state is R1 and all attack vectors for state pair (R2, LO) are removed. Then, it merges the states of all remaining attack vectors with the same SD-URL and attack class into a single attack vector that distinguises S_t from $n \geq 2$ other states. In our example, the attack vectors do not merge further. Next, it initializes a set P with all pairs of states and browsers to be distinguished (Line 5). The algorithm goes into a loop that at each iteration it identifies the attack vector that covers most remaining pairs in P (Lines 6-14). The loop iterates until all pairs have been covered, no attack vectors remain, or the remaining attack vectors do not allow distinguishing the remaining pairs. To select an attack vector, a `score` function is used that assigns higher scores to attack vectors that cover more pairs in P, penalizing attack classes that may interfere with other vectors (Line 7). For example, an *EventsFired* attack vector using the `script` tag may trigger CSP violation reports that interfere with a CSP policy for *CSPViolation* that targets script resources. If the score is zero, the loop breaks as the remaining attack vectors do not allow distinguishing the remaining pairs. Otherwise, the selected attack vector is appended to the output (Line 10), the newly covered pairs are removed from P (Line 11), and the attack vector is removed from the available list (Line 12).

In our example, the first loop iteration selects the attack vector ({LO}, `offline.php`, *EF-XctoObject*) as it covers three pairs, differentiating the logout state for Chrome, Firefox and Edge. The next loop iteration selects the attack vector ({R2}, `api.php`, *EF-StatusErrorObject*) as it covers two other pairs, differentiating all remaining states for Firefox and Edge. Finally, the last iteration chooses ({R2}, `api.php`, *EF-StatusErrorLink*) which covers the remaining state for Chrome. At that point, no more pairs remain to be covered, and the algorithm outputs the selected attack vectors. The algorithm also outputs the pair set P. If empty, the attack page distinguishes the target state from all other states for all target browsers. Otherwise, some states may not be distinguishable for some target browsers.

For each attack class, the attack page generation module has a template to implement the attack. For each selected attack vector, it chooses one inclusion method in the attack class, and applies the corresponding template with the SD-URL. All instantiated templates are integrated into the output attack page.

V. Ethics

Our experiments do not target any real user of the live sites. All testing on live sites is restricted to user accounts that we created on those sites exclusively for this purpose. The process of validating that the attacks found on open-source web applications work on live installations of those applications is similarly restricted to accounts owned by the authors. The impact on live sites is limited to receiving a few thousand requests for valid resources in the site. We take two actions to limit the load on live sites from our testing. First, we spread the requests over time to avoid spike loads. Second, we disable the timing XS-Leak in our experiments, which requires sending hundreds, or even thousands, of requests per SD-URL, generating the highest load.

We have disclosed our attacks to the four web applications, receiving confirmation of the issues from HotCRP, GitLab, and GitHub, while OpenCart has not replied. The disclosure process for the web sites is ongoing. All reported attacks have been confirmed and some attacks have already been patched (e.g., HotCRP, linkedin.com). We avoid providing SD-URLs for attacks not yet patched. We have also reported our results to the three browser vendors, as well as the Tor project. We incorporate their feedback into our defenses discussion in Section VII.

We acknowledge that publicly releasing Basta-COSI makes it possible for attackers to misuse it to find COSI attacks. However, we argue that this applies to any penetration testing and vulnerability discovery tool (open source or commercial). Other distribution models such as Software-as-a-Service could potentially mitigate this risk, but would also limit the usefulness for the research community. We believe determined attackers will still find a way to attack sites even without Basta-COSI. Thus, we favor the benefit for defenders and the research community.

VI. Experiments

This section presents the evaluation of Basta-COSI on four open source web applications (HotCRP, GitLab, GitHub Enterprise, OpenCart) and the 58 web sites in the Alexa Top 150 [15] where we could create user accounts. These targets are popular, allow us to test on white-box (open source) and black-box (deployed) scenarios, and cover services with multiple user states. Section VI-A describes the results on Web applications, Section VI-B on Alexa web sites, and Section VI-C details some attacks found.

A. Evaluation on Web Applications

Table V summarizes the results of applying Basta-COSI on the four web applications we installed locally. It details the results for each tool module, as well as the COSI attacks found. The data collection part shows the number of input state scripts provided to Basta-COSI, the number of URLs crawled, and the number of SD-URLs identified. The attack vector identification part shows the total number of attack vectors identified, the number of state pairs they cover, and the number of XS-Leaks they use. The attack page generation part shows the number of states uniquely distinguished (UD) from other states, the number of states partially distinguished (PD) excluding UD states, and the minimum/average/maximum attack vectors in the attack pages. Finally, the attacks found part shows the type and browsers affected for the identified attacks.

Depending on the target, we created 3–6 state scripts to use Basta-COSI. One script always corresponds to the logged out (LO) state and the others are target-specific. For example, for GitLab the other 5 states are for maintainer, developer, reporter, guest (read-only access), and a user with no read access to the repository. Like a fuzzing tool, Basta-COSI will try to find attacks until the allocated time budget runs out. We let Basta-COSI run for a maximum of 24 hours on each target, although after a few hours the crawling typically does not find any new URLs. The data collection results show that SD-URLs are very common, on average 68% of the discovered URLs are SD-URLs (and up to 99% in GitHub).

Basta-COSI finds between 58 and 992 attack vectors in each target using up to 3 XS-Leaks. The results show that on average the generated attack pages use more than one attack vector. Account type and deanonymization attacks always require multiple vectors, while login detection is oftentimes possible with a single vector. This highlights the importance of our approach to combine attack vectors in order to handle more than two states and multiple browsers. Some states can be uniquely identified, i.e., distinguished from any other state, and the rest can be partially distinguished. We found no state that could not be distinguished at all. It is important to note that partially distinguishable states can also be used in attacks. For example, not being able to differentiate the administrator from a normal user does not matter if the administrator is not targeted by the attack, i.e., not sent the attack page URL. Overall, Basta-COSI finds attacks on all four applications: login detection attacks on all four, deanonymization attacks on three, and account type identification on two.

B. Evaluation on Web Sites

We test sites from the Alexa Top 150 that are not duplicates (e.g., amazon.com vs. amazon.de) and where we could create

Target	Data Collection			Attack Vector Identification			Attack Page Generation					Attacks Found			
			SD		State		UD	PD	Vectors			Login	Account		Access
	States	URLs	URLs	Vectors	Pairs	XS-Leaks	States	States	Min	Avg	Max	Detection	Type	Deanon.	Detection
HotCRP	5	68	65	116	7	3	1	4	1	1.6	3	C,E,F	-	C,E,F	-
GitLab	6	52	19	236	14	1	2	4	1	1.9	2	C,E,F	C,E,F	C,E,F	-
GitHub	4	91	90	992	6	1	4	0	1	1.8	2	C,E,F	C,E,F	C,E,F	-
OpenCart	5	51	32	72	7	1	2	3	1	1.1	2	C,E,F	-	-	-

TABLE V: Basta-COSI evaluation results. For every target application and site, it shows the data for each tool module, as well as the type and browsers affected for the attacks found. Browsers are abbreviated as Chrome (C), Firefox (F), and Edge (E).

Attack Type	Tested	Vulnerable
Login Detection	58	58
Deanonymization	58	36
SSO Status	12	12
Access Detection	11	5
Account Type	3	3
Total Sites	58	58

TABLE VI: Web sites vulnerable to each attack type

Attack	Br	EF	OP	PM	CSS	JSE	JOR	CSP	ACE
Login Detect.	C	2457	2532	9	0	2	0	885	63
	F	1587	1511	9	0	2	0	424	0
	E	676	1286	9	0	2	0	434	0
Account Type	C	175	82	0	0	0	0	126	3
	F	173	85	0	0	0	0	2	0
	E	39	36	0	0	0	0	12	0
Deanon.	C	644	546	2	0	0	0	31	17
	F	447	420	2	0	0	0	79	0
	E	201	288	2	0	0	0	81	0
Access Detect.	C	98	12	0	0	0	72	0	0
	F	1	10	0	0	0	0	0	0
	E	3	10	0	0	0	0	0	0
SSO Status	C	0	0	0	0	0	0	12	0
	F	0	0	0	0	0	0	12	0
	E	0	0	0	0	0	0	12	0

(**Legend:** EF=EventFire; OP=ObjectProperties; PM=PostMessage; CSS= CSSPropRead; JSE=JSError; JOR=JSObjectRead; CSP=CSPViolation; ACE=AppCacheError)

TABLE VII: Attack vectors found per XS-Leak and browser.

free accounts. This excludes sites without user accounts, that required a phone number in a specific area, or that demanded credit card information. This leaves us with 58 sites, of which only 12 support SSO, and only 3 have multiple types of free accounts (excluding the administrator account that we cannot obviously create). For access detection, we focus on privacy sensitive sites, more specifically adult sites, on the Alexa Top 150, regardless if they have user accounts.

Table VI summarizes the number of tested and vulnerable sites for each attack type. For login detection, SSO status, and account type identification, Basta-COSI discovers XS-Leaks against all tested sites. In addition, it finds deanonymization attacks in 57% of the sites and access detection attacks in 45%. The results show that login detection attacks are easiest to find, but that by combining multiple attack vectors it is possible to find more powerful attacks targeting more than two states in 72% of the sites. Regarding false positives, we rarely observed them in two situations. One was due to Basta-COSI waiting 6 seconds to collect events and some pages being slower to load. The other one was when Basta-COSI sent too many queries and a site started replying with CAPTCHAs. We expect that increasing the timeout and distributing the queries over multiple IPs would eliminate those false positives. We do not evaluate false negatives, as we lack ground truth of the COSI attacks present in the targets. However, we acknowledge that, like any testing tool, false negatives are possible, e.g., Basta-COSI can only find COSI attacks that are instances of the 40 attack classes it supports.

The support in Basta-COSI for multiple XS-Leaks and multiple browsers allows to compare the prevalence of the XS-Leaks, as well as the attack surface of the browsers, on the same set of SD-URLs, i.e., independently of the crawler's coverage. Table VII details the distribution of attack vectors per XS-Leak for each attack type and browser pair. XS-Leak prevalence widely varies. Most attack vectors use *EventsFired*, *Object Properties*, and *CSPViolation* XS-Leaks. Our novel *postMessage* XS-Leak ranks sixth out of eight XS-Leaks, producing attack vectors on 11 different sites including blogger.com, ebay.com, reddit.com, and youtube.com. The least prevalent XS-Leak is *CSSPropRead* for which Basta-COSI does not find any attack vector, showing that SD-URLs on CSS content that leak user state are not common. The comparison also shows that Chrome has a larger attack surface, ranking first in number of attack vectors in all eight XS-Leaks.

C. Example Attacks

This section details some of the attacks Basta-COSI found that involve more than two states. All attacks work on the three tested browsers, unless specifically noted.

HotCRP. Basta-COSI found an attack for determining whether the victim is a reviewer of a specific paper, which we have used as running example. The attack page (Listing 1) uses three attack vectors, one for login detection on all three browsers, and two (one for Chrome and another for Firefox/Edge) to identify if the victim submitted a review for the target paper. To launch the attack, the attacker collects the email addresses of the program committee members and sends them a spear-phishing email to convince them to click on the attack page URL. Since the attack was found on a local HotCRP installation, to test it on conferences hosted at hotcrp.com, we had to update the SD-URLs with the proper domain and conference name. We verified the attack and reported it to the HotCRP developer, who confirmed the issue and has released a patch [10].

GitLab and GitHub. Attacks are found in both GitLab and github.com that allow determining if the victim is the owner of a repository (or of a snippet). Both attacks first use a login detection attack. If the victim is logged in, the attack page uses an *EventFire* attack class using a SD-URL for editing the repository settings (or the snippet) to detect if the victim has administrative rights. For GitHub Enterprise installations, another attack allows distinguishing the administrator from other users by including the URL for accessing staff tools.

LinkedIn. A *CSPViolation* attack allows distinguishing the account type (free or premium) using the SD-URL https://www.linkedin.com/cap/. This attack has already been fixed following our disclosure. A second attack allows determining if the victim owns a specific LinkedIn profile using the *OP-WindowProperties* attack class. The underlying cause of this attack is that the number of frames in a LinkedIn profile page is 3 when visited by the owner of the profile, and 4 otherwise.

Blogger. Multiple deanonymization attacks are found for determining if the victim is the owner of a specific blog. The attacker needs to know the *blogID* of the target victim, which can be found on the HTML source of the target blog. The attacks combine a *CSPViolation* login detection attack vector with another deanonymization attack vector from different attack classes (e.g., *postMessage*, *EF-CtMismatchScript*). This shows how attacks can combine multiple attack vectors using different XS-Leaks, highlighting the value of our generic approach not being specific to any XS-Leak.

IMDB. A deanonymization attack allows determining if the victim owns a specific IMDB account using a SD-URL that contains the user identifier. This attack can determine if the visitor is a specific person from the film industry by including the user identifier obtained from the profile for that person.

Amazon. *CSPViolation* attacks are found that leak if the victim is using the Amazon Kindle Direct Publishing (KDP) service, or has accepted the KDP terms and policies. That information could be used for targeted advertising, e.g., to show advertisements of kindle books to the victim.

Pornhub. Attacks are found using the *OP-Window-Properties* and *OP-FrameCount* for determining if the victim is the owner of a specific username, thus enabling deanonymization of the account in a closed-world setting. The underlying reason for the *OP-FrameCount* attack is similar to that of the LinkedIn attack, but mounted on Pornhub's playlist URLs.

Pinterest. A *CSPViolation* attack can be mounted with the Facebook SSO initiation URL for determining whether the victim authenticated into Pinterest using its Facebook account. A similar attack was found for Google's SSO.

Imgur. An attack based on *EF-StatusErrorScript* can be used to determine if the victim uploaded an image (e.g., copyrighted, taken without permission) to this image sharing site. The vendor has awarded us a bug bounty for this report [46].

VII. Defenses Against COSI Attacks

This section discusses existing and upcoming defenses against COSI attacks.

SameSite cookies. COSI attacks leverage the automatic inclusion of HTTP cookies [19], client-side certificates [45], and HTTP Authentication credentials [30] in requests sent by web browsers, known as the ambient authority problem in browsers [25]. Web sites can use the `SameSite` attribute in a `Cookie` header to prevent the browser from sending that cookie in cross-site requests [43], [67]. This defense disables SD-URLs whose responses are based on states saved in cookies. On the other hand, it does not prevent leakage by HTTP Authentication credentials and client-side certificates, it needs to be set for each cookie; it may be challenging to deploy in web sites with legitimate cross-origin requests [58]; and its implementation in browsers can have flaws [29]. When we disclosed our results to the browser vendors, we were told they plan to address COSI attacks by marking all cookies by default as `SameSite=Lax`, unless the site specifically disables them with `SameSite=None`, or makes it stricter with `SameSite=Strict` [69]. This change is already planned for Chrome [11] and Firefox [12]. However, this defense will initially ship behind a configuration option since it may affect functionality that requires cross-origin requests.

Session-specific URLs. Web sites can use URLs that include a session-specific, non-guessable, token. The token must be cryptographically bound to the session identifier (e.g., the hash of the identifier), and the web site must verify this relationship for all HTTP requests. Session-specific URLs prevent the attacker from identifying SD-URLs for the victim's session, avoiding COSI attacks. This defense does not depend on browser vendors and can be deployed right away. On the other hand, it can be costly to deploy, increases complexity, may impact performance, and the web site must ensure that the tokens cannot be leaked or brute forced [25].

Cross-Origin-Resource-Policy. An emerging HTTP response header that allows web sites to ask browsers to disallow cross-origin requests to specific resources [2]. The request is not prevented, rather the browser avoids leakage by stripping the response body. Currently supported by Chrome and Safari.

Fetch metadata. An emerging set of HTTP request headers that send additional provenance data about a request [68], e.g., the HTML element triggering a cross-site request. Currently supported by Chrome. A web site can use this information to design policies that block potentially malicious requests. e.g., inclusion of a non-image resource with an *img* tag.

Cross-Origin-Opener-Policy. There is ongoing discussion on a new HTTP response header to prevent malicious web sites from abusing other web sites by opening them in a window [3]. This defense could protect against COSI attack classes that use the *window.open* inclusion method (e.g. *OP-Window Properties*, *postMessage*).

Tor Browser. The Tor Browser takes preventive measures against timing-based COSI attacks [54]. Additionally, it isolates the browser's state based on the URL in the address bar. Therefore, it does not attach cookies and `Authorization` header values to cross-origin HTTP requests generated by inclusions using HTML tags. However, the state isolation is not enforced for the *window.open* method, so authentication headers are still attached to HTTP requests generated using this inclusion method. Therefore, Tor Browser users are still vulnerable to *OP-WindowProperties* and the new postMessage attack class we discovered.

SD-URL patching. When reporting our attacks, we mentioned SameSite cookies as a good defense in terms of protection, since it tackles the root cause of COSI attacks, and cost to deploy. However, the developers that already patched our attacks did not take that suggestion and instead applied a fix specific to the reported SD-URLs. For example, the HotCRP developer mentioned that SameSite cookies is not available in PHP until PHP 7.3, and instead modified the code to always return a 200 HTTP status code with JSON content. This fixes our attack, but it will not fix future attacks on other status codes and content types. In another example, LinkedIn patched our reported user deanonymization *OP-FrameCount* attack by making sure that the reported SD-URL returned the same number of frames for all users. These examples show that developers currently consider URL-specific fixes a quick solution, despite its lack of generality.

VIII. Discussion

This section discusses limitations of our approach and possible future improvements.

Preparation overhead. To use Basta-COSI, the tester first needs to create accounts at the target site and provide state scripts that use those accounts. Similar overhead is required by other web security testing tools, when they need to examine the logged in parts of a web site. Furthermore, Basta-COSI is designed for web site administrators to test their own sites. We believe the cost of creating test accounts for your own site is a reasonable one-time effort, as these accounts can then be reused for other tests. In fact, we expect many sites to already have such test accounts in place for other types of testing.

Support for other browsers. Basta-COSI currently supports the three most popular browsers: Chrome, Firefox, and Edge. We did not include support for Safari because we run our experiments on Windows and Apple stopped releasing Safari for Windows in 2012. Adding support for other browsers is a matter of additional engineering work. Of particular interest would be adding support for mobile platform browsers given their popularity and that COSI attacks on those browsers have been little explored. Support for mobile browsers in Basta-COSI could be achieved by integrating a mobile testing platform, e.g., Appium [1].

Support for other crawlers. Basta-COSI uses ZAP's Spider module [7] for crawling the target site. The coverage of this crawler may be limited on JavaScript-intensive web sites. It is likely that some SD-URLs were not discovered by the crawler for this reason, which may have caused COSI attacks to go unnoticed. Basta-COSI's modular design should easily allow to integrate other crawlers to increase coverage. Still, despite the potentially limited crawling, Basta-COSI was able to find COSI attacks in all tested targets.

Dynamic page element detection. To identify SD-URLs, Basta-COSI removes dynamic page elements from HTTP responses. Our detection of some dynamic page elements, e.g., CSRF tokens, is based on heuristics that could introduce errors. However, there are a couple of mitigating reasons, which may explain why we did not observe such errors in our testing. First, even if a URL is wrongly identified as a SD-URL, Basta-COSI may later discard it as non-exploitable. Second, dynamic elements often do not impact the leak methods (e.g., events fired, properties read).

Timing. Basta-COSI supports the timing XS-Leak through the video parsing technique described in [64]. However, we did not use the timing XS-Leak in our experiments, which may have prevented Basta-COSI from finding further attacks. The main reason for disabling the timing XS-Leak is that in order to attain the same level of reliability as other attack classes, it requires sending hundreds [64], or even thousands [13], of HTTP requests per SD-URL. This increases the load at the target and causes some web sites to respond with defenses (e.g., CAPTCHAs, blocking) that hamper the testing. We noticed this initially on linkedin.com. In addition to the high load, we observed another three challenges in using the timing XS-Leak. First, we cannot generalize a timing attack. With timing, we always need to measure the timing for each URL in the target site; we cannot reuse what we learn from one attack in new attacks. Second, timing information is harder to use as the number of states increases. For example, if a URL allows downloading a file only to its owner, there may not be a clear timing difference between an unauthenticated user and an authenticated one that is not the owner. Finally, it is hard to combine in the same attack timing with the non-timing XS-Leaks. Due to these challenges by default Basta-COSI does not use the timing XS-Leak. We leave applying timing leaks to more than two states for future work.

Discovering new XS-Leaks. We have systematically explored existing COSI attacks and the XS-Leaks they use, generalizing them into COSI attack classes. In this process, we have discovered a novel *postMessage* XS-Leak. However, it is very likely that there exist more, currently unknown, XS-Leaks leveraging other browser APIs. Systematically exploring the browser API surface to identify all possible XS-Leaks remains an open challenge, which we plan to explore in future work.

IX. Related Work

Prior COSI attack instances. Table VIII summarizes the 23 prior works proposing COSI attack instances we have identified. The first instance of a COSI attack was proposed in 2006 by Grossman and Hansen [36]. It was a login detection attack using the *img* tag and the EventsFired XS-Leak (*EF-CtMismatchImg* attack class). Since then, EventFired attacks have been shown to apply to other HTML tags and content types [22], [34], [35], [65]. Recently, Staicu and Pradel [61] showed that EventsFired attacks can be combined with shareable images to deanonymize users of image sharing services.

In another blog post in 2006, Grossman [33] introduced the first instance of the *JSError* attack class that leverages the type and line number of errors triggered when a JavaScript resource is included using the *script* tag. This attack was then demonstrated on popular sites like Amazon [59]. Inspired by Grossman's attacks, Evans [26] presented the first instance of the *CSSPropRead* attack class, leveraging the presence of certain objects and variables from an included JS resource. In a 2012 post Grossman presented multiple attack instances including the first instances of the *JSObjectRead* attack class and the first attack using the readable object properties XS-Leak [35]. Lekies et al. [50] extended the *JSObjectRead* class with more techniques such as prototype tampering and showed that

Reference	Year	Type	Attack Classes	Browsers
Grossman & Hansen [36]	2006	Blog	EF-CtMismatchImg	-
Grossman [33]	2006	Blog	JSError	F
Shiflett [59]	2006	Blog	JSError	F
Bortz et al. [21]	2007	Paper	Timing	F, S
Grossman [34]	2008	Blog	EF-CtMismatchScript, EF-CtMismatchImg	F
Evans [26]	2008	Blog	CSSPropRead	F
Evans [27]	2009	Blog	Timing	-
Cardwell [22]	2011	Blog	EF-StatusErrorScript, EF-CtMismatchImg	C, F, IE
Grossman [35]	2012	Blog	EF-StatusErrorIFrame, EF-CtMismatchScript, OP-LinkSheet, OP-FrameCount, EF-CtMismatchImg, JSObjectRead	F
Homakov [40]	2013	Bug	CSPViolation	C, F, IE
Gelernter & Herzberg [31]	2015	Paper	Timing	-
Goethem et al. [64]	2015	Paper	Timing, EF-CtMismatchVideo	C
Lekies et al. [50]	2015	Paper	JSObjectRead	C
Lee et al. [47]	2015	Paper	AppCacheError	C
Schwenk et al. [57]	2017	Paper	OP-LinkSheet	IE, E
Masas [52]	2018	Blog	OP-WindowProperties	C
Yoneuchi [71]	2018	Blog	CSPViolation	F
Gulyas et al. [38]	2018	Paper	CSPViolation	C
Acar [14]	2018	Paper	OP-MediaStatus	C, F
Staicu & Pradel [61]	2019	Paper	EF-CtMismatchImg	C, F
Masas [55]	2019	Blog	OP-WindowProperties	C
Sanchez et al. [56]	2019	Paper	Timing	C
XSLeaks [65]	2019	Project	EF-CtMismatchImg, OP-FrameCount, CSPViolation, Timing, EF-CtMismatchObject, OP-ImgDimension, OP-MediaDuration, OP-WindowProperties, EF-CacheLoadCheck	C, F, E

(**Legend:** F=Firefox; S=Safari; C=Chrome; IE=Internet Explorer; E=Edge; -= we couldn't find a browser mentioned in the article)

TABLE VIII: Summary of previously proposed COSI attacks

JSObjectRead attacks can be defended by making the URLs of script files unpredictable and including JS parser-breaking strings in dynamic JS files. After Grossman's initial attack using the FrameCount readable object property, instances of attack classes leveraging other properties (e.g., window frame count, width, height, duration, cssRules, media error) have been proposed [14], [52], [55], [57], [65].

Homakov [40], [41] showed that cross-origin and sub-domain redirections can be detected by abusing CSP. This approach has been used for login detection and fingerprinting attacks [38], [71]. Lee et al. showed that the AppCache feature can be abused to differentiate between 200 status responses and redirection or error responses [47]. Recently, Staicu et al. [61] showed that a deanonymization attack can be mounted using images uploaded to GitHub. We generalized this attack on GitHub also to non-image resources. Bortz et al. [21] showed that the timing of the events fired when a resource is loaded using the *img* HTML tag is a good metric to determine the state of a user at a target site. Evans [27] and Gelernter and Herzberg [31] applied similar approaches for mounting cross-site search attacks. Goethem et al. [64] showed that the parsing time of the included resources is a better alternative and that the `Referer` and `Origin` headers can help preventing such attacks. Recently, Sanchez et al. [56] have measured the scale of timing-based login and access detection attacks.

This work shows that the above are all instances of COSI attacks, and demonstrates how to build complex COSI attacks that handle more than two states and multiple browsers.

Browser history sniffing attacks. Multiple works have studied history sniffing attacks that use browser side channels to determine whether a user has accessed certain web sites [23], [28], [53], [60], [70]. To defend against history sniffing attacks Jackson et al. proposed to increase the isolation of different origins [42] and Wondracek et al. proposed adding non-predictable tokens in URLs and using the POST method [70]. History sniffing attacks are similar to COSI attacks in leveraging a browser side channel, but fundamentally differ in the absence of a target site and in that the attack page does not send cross-origin requests.

Attacks using postMessage. Guan et al. [37] analyzed privacy issues in postMessages broadcasted by popular web sites and Stock et al. showed that usage of broadcasted postMessages has been increasing [62]. Our postMessage XS-Leak leverages differences between broadcasted postMessages in SD-URLs and does not require that messages contain sensitive data.

X. Conclusion

We have presented COSI attacks as a comprehensive category and have introduced a novel approach to identify and build complex COSI attacks that differentiate more than two states and support multiple browsers. Our approach combines multiple attack vectors, possibly using different XS-Leaks. To enable our approach, we have introduced the concept of COSI attack classes and have proposed novel techniques to discover attack classes from existing instances of COSI attacks. In this process, we have discovered a novel browser XS-Leak based on *window.postMessage*. We have implemented our approach into Basta-COSI, a tool to find COSI attacks in a target web site. We have applied Basta-COSI to test four stand-alone web applications and 58 popular web sites, finding COSI attacks against each of them.

Acknowledgments

We thank Adam Doupe and the anonymous reviewers for their insightful comments and feedback. This research was largely performed while Soheil Khodayari was an intern at the IMDEA Software Institute. This research has received funding from the European Union Horizon 2020 Research and Innovation Programme under the ELASTEST Grant Agreement No. 731535. This work was also supported by the Regional Government of Madrid through the BLOQUES-CM grant P2018/TCS-4339 and by the Spanish Government through the SCUM grant RTI2018-102043-B-I00. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators, and do not necessarily reflect the views of the sponsors.

References

[1] Appium: Mobile app automation made awesome. [Online]. Available: https://appium.io/

[2] Cross-Origin-Resource-Policy (was: From-Origin). [Online]. Available: https://github.com/whatwg/fetch/issues/687

[3] ‘Cross-Origin-Window-Policy‘ header. [Online]. Available: https://github.com/whatwg/html/issues/3740

[4] Elastest: An elastic platform to ease end to end testing. [Online]. Available: https://elastest.eu/

[5] OWASP Zed Attack Proxy. [Online]. Available: https://www.owasp.org/index.php/ZAP

[6] Selenium-python. [Online]. Available: https://selenium-python.readthedocs.io/index.html

[7] Spider. [Online]. Available: https://github.com/zaproxy/zap-core-help/wiki/HelpStartConceptsSpider

[8] Web Video Text Tracks Format (WebVTT). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebVTT_API

[9] (2014) Using the application cache. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache

[10] (2019) Attempt to plug an information leak represented by http status. [Online]. Available: https://github.com/kohler/hotcrp/commit/406a966aad00a762460fbc62cfb04a7532fc9fbd

[11] (2019) Intent to Implement and Ship: Cookies with SameSite by default. [Online]. Available: https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/AknSSyQTGYs/SSB1rTEkBgAJ

[12] (2019) Intent to implement: Cookie SameSite=lax by default and SameSite=none only if secure. [Online]. Available: https://groups.google.com/forum/#!msg/mozilla.dev.platform/nx2uP0CzA9k/BNVPWDHsAQAJ

[13] (2019) Timing Attacks using Machine Learning. [Online]. Available: https://parzelsec.de/timing-attacks-with-machine-learning/

[14] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, “Web-based attacks to discover and control local iot devices,” in Proceedings of the Workshop on IoT Security and Privacy, 2018.

[15] Amazon. The top 500 sites on the web. [Online]. Available: https://www.alexa.com/topsites

[16] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra, “Formal analysis of saml 2.0 web browser single sign-on: Breaking the saml-based single sign-on for google apps,” in Proceedings of the ACM Workshop on Formal Methods in Security Engineering, 2008.

[17] C. Bansal, K. Bhargavan, and S. Maffeis, “Discovering concrete attacks on website authorization by formal analysis,” in Proceedings of the IEEE Computer Security Foundations Symposium, 2012.

[18] A. Barth, “The web origin concept,” 2010. [Online]. Available: https://tools.ietf.org/html/rfc6454

[19] ——, “Http state management mechanism,” 2011. [Online]. Available: https://tools.ietf.org/html/rfc6265

[20] A. Barth, C. Jackson, and J. C. Mitchell, “Robust defenses for cross-site request forgery,” in Proceedings of the ACM Conference on Computer and Communications Security, 2008.

[21] A. Bortz, D. Boneh, and N. Palash, “Exposing private information by timing web applications,” in Proceedings of the International Conference on World Wide Web, 2007.

[22] M. Cardwell. (2011) Abusing HTTP Status Codes to Expose Private Information. [Online]. Available: https://www.grepular.com/

[23] A. Clover, “Css visited pages disclosure,” BUGTRAQ mailing list posting, 2002.

[24] G. Crawley. (2018) Thousands hit by porn blackmail scam. [Online]. Available: https://www.express.co.uk/news/uk/993251/porn-blackmail-scam-cyber-criminals-demanding-ransom

[25] A. Czeskis, A. Moshchuk, T. Kohno, and H. Wang, “Lightweight server support for browser-based csrf protection,” in Proceedings of the International Conference on World Wide Web, 2013.

[26] C. Evans. (2008) Cross-domain leaks of site logins. [Online]. Available: https://scarybeastsecurity.blogspot.com/2008/08/cross-domain-leaks-of-site-logins.html

[27] ——. (2009) Cross-domain search timing. [Online]. Available: https://scarybeastsecurity.blogspot.com/2009/12/cross-domain-search-timing.html

[28] E. W. Felten and M. A. Schneider, “Timing attacks on web privacy,” in Proceedings of the ACM Conference on Computer and Communications Security, 2000.

[29] G. Franken, T. V. Goethem, and W. Joosen, “Who left open the cookie jar? a comprehensive evaluation of third-party cookie policies,” in Proceedings of the USENIX Security Symposium, 2018.

[30] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, “Http authentication: Basic and digest access authentication,” 1999. [Online]. Available: https://tools.ietf.org/html/rfc2617

[31] N. Gelernter and A. Herzberg, “Cross-site search attacks,” in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2015.

[32] J. Grossman. (2006) Advanced Web Attack Techniques using GMail. [Online]. Available: http://blog.jeremiahgrossman.com/2006/01/advanced-web-attack-techniques-using.html

[33] ——. (2006) I know if you’re logged-in, anywhere. [Online]. Available: https://blog.jeremiahgrossman.com/2006/12/i-know-if-youre-logged-in-anywhere.html

[34] ——. (2008) Login Detection, whose problem is it? [Online]. Available: https://blog.jeremiahgrossman.com/2008/03/login-detection-whose-problem-is-it.html

[35] ——. (2012) I Know What Websites You Are Logged-In To (Login-Detection via CSRF). [Online]. Available: http://web.archive.org/web/20160317054027/https://www.whitehatsec.com/blog/i-know-what-websites-you-are-logged-in-to-login-detection-via-csrf/

[36] J. Grossman and R. Hansen. (2006) Detecting States of Authentication With Protected Images. [Online]. Available: http://web.archive.org/web/20150417095319/http://ha.ckers.org/blog/20061108/detecting-states-of-authentication-with-protected-images/

[37] C. Guan, K. Sun, Z. Wang, and W. Zhu, “Privacy breach by exploiting postmessage in html5: Identification, evaluation, and countermeasure,” in Proceedings of the ACM on Asia Conference on Computer and Communications Security, 2016.

[38] G. G. Gulyas, D. F. Some, N. Bielova, and C. Castelluccia, “To extend or not to extend: on the uniqueness of browser extensions and web logins,” in Workshop on Privacy in the Electronic Society, 2018.

[39] A. Hern. (2016) Spouses of ashley madison users targeted with blackmail letters. [Online]. Available: https://www.theguardian.com/technology/2016/mar/03/ashley-madison-users-spouses-targeted-by-blackmailers

[40] E. Homakov. (2013) Bug 313737 - Disclose domain of redirect destination taking adventadge of CSP. [Online]. Available: https://bugs.chromium.org/p/chromium/issues/detail?id=313737

[41] ——. (2014) Using Content-Security-Policy for Evil. [Online]. Available: http://homakov.blogspot.com/2014/01/using-content-security-policy-for-evil.html

[42] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell, “Protecting browser state from web privacy attacks,” in Proceedings of the International Conference on World Wide Web, 2006.

[43] A. Janc and M. West, “How do we Stop Spilling the Beans Across Origins,” 2018. [Online]. Available: https://www.arturjanc.com/cross-origin-infoleaks.pdf

[44] M. A. Jaro, “Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida,” Journal of the American Statistical Association, vol. 84, no. 406, pp. 414–420, 1989.

[45] M. Johns and J. Winter, “RequestRodeo: Client side protection against session riding,” 2006. [Online]. Available: https://www.owasp.org/images/4/42/RequestRodeo-MartinJohns.pdf

[46] S. Khodayari. (2019) De-anonymization attack: Cross site information leakage. [Online]. Available: https://hackerone.com/reports/723175

[47] S. Lee, H. Kim, and J. Kim, “Identifying cross-origin resource status using application cache,” in Proceedings of the Network and Distributed Systems Security Symposium, 2015.

[48] S. Lekies, M. Johns, W. Tighzert et al., “The state of the cross-domain nation,” in Proceedings of the IEEE Web 2.0 Security & Privacy, 2011.

[49] S. Lekies, B. Stock, and M. Johns, "25 million flows later: large-scale detection of dom-based xss," in Proceedings of the ACM SIGSAC conference on Computer &#38; communications security, 2013.

[50] S. Lekies, B. Stock, M. Wentzel, and M. Johns, "The unexpected dangers of dynamic javascript," in Proceedings of the USENIX Security Symposium, 2015.

[51] R. Linus. (2016) Your Social Media Fingerprint. [Online]. Available: https://github.com/RobinLinus/socialmedia-leak

[52] R. Masas. (2018) Patched Facebook Vulnerability Could Have Exposed Private Information About You and Your Friends. [Online]. Available: https://www.imperva.com/blog/facebook-privacy-bug/

[53] L. Olejnik, C. Castelluccia, and A. Janc, "Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns," in Proceedings of the Workshop on Hot Topics in Privacy Enhancing Technologies, 2012.

[54] M. Perry, E. Clark, S. Murdoch, and G. Koppen, "The Design and Implementation of the Tor Browser [DRAFT]," 2018. [Online]. Available: https://2019.www.torproject.org/projects/torbrowser/design/#identifier-linkability

[55] Ron, Masas. (2019) Mapping communication between facebook accounts using a browser-based side channel attack. [Online]. Available: https://www.imperva.com/blog/mapping-communication-between-facebook-accounts-using-a-browser-based-side-channel-attack/

[56] I. Sanchez-Rola, D. Balzarotti, and I. Santos, "Bakingtimer: Privacy analysis of server-side request processing time," in Proceedings of the Annual Computer Security Applications Conference, 2019.

[57] J. Schwenk, M. Niemietz, and C. Mainka, "Same-origin policy: Evaluation in modern browsers," in Proceedings of the USENIX Security Symposium (USENIX Security 17), 2017.

[58] R. Sharma, "Preventing cross-site attacks using same-site cookies," 2017. [Online]. Available: https://blogs.dropbox.com/tech/2017/03/preventing-cross-site-attacks-using-same-site-cookies/

[59] C. Shiflett. (2006) Javascript Login Check. [Online]. Available: http://shiflett.org/blog/2006/javascript-login-check

[60] M. Smith, C. Disselkoen, S. Narayan, F. Brown, and D. Stefan, "Browser history re:visited," in Proceedings of the USENIX Workshop on Offensive Technologies, 2018.

[61] C. A. Staicu and M. Pradel, "Leaky images: Targeted privacy attacks in the web," in Proceedings of the USENIX Security Symposium, 2019.

[62] B. Stock, M. Johns, M. Steffens, and M. Backes, "How the web tangled itself: Uncovering the history of client-side web (in)security," in Proceedings of the USENIX Security Symposium, 2017.

[63] T. Terada, "Identifier based XSSI attacks," 2015. [Online]. Available: https://www.mbsd.jp/Whitepaper/xssi.pdf

[64] T. Van Goethem, W. Joosen, and N. Nikiforakis, "The clock is still ticking: Timing attacks in the modern web," in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2015.

[65] E. Vela Nava, L. Herrera, R. Masas, K. Kotowicz, A. Saftnes, Terjanq, and Stephen. (2019) Browser Side Channels. [Online]. Available: https://github.com/xsleaks/xsleaks/wiki/Browser-Side-Channels

[66] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through facebook and google: a traffic-guided security study of commercially deployed single-sign-on web services," in Proceedings of the IEEE Symposium on Security and Privacy, 2012.

[67] M. West, "Same-site cookies," 2016. [Online]. Available: https://tools.ietf.org/html/draft-west-first-party-cookies-07

[68] ——, "Fetch metadata request headers," 2018. [Online]. Available: https://mikewest.github.io/sec-metadata/

[69] ——, "Incrementally better cookies," 2019. [Online]. Available: https://tools.ietf.org/html/draft-west-cookie-incrementalism-00

[70] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to de-anonymize social network users," in Proceedings of the IEEE Symposium on Security and Privacy, 2010.

[71] T. Yoneuchi. (2018) Detect the Same-Origin Redirection with a bug in Firefox's CSP Implementation. [Online]. Available: https://diary.shift-js.info/csp-fingerprinting/

[72] M. Zalewski. (2008) Browser security handbook, part 2. [Online]. Available: https://code.google.com/archive/p/browsersec/wikis/Part2.wiki#Same-origin_policy