Privacy - Canvas Poisonning

Daussy Fabio, Mauger Samuel, Meyran Hilan

May 5, 2024

Contents

Abstract

The following document is not the final report of our project but is useful to have a preview of what we prepared during these three weeks.

1 Introduction

For this project, we read the FP-Scanner paper [VLRR18] and focused on the Canvas fingerprinting part. Our objectives are to understand how we can detect the usage of Canvas Defender, a web browser extension used to protect Canvas fingerprint gathering. Then, once the plug-in is detected, try to erase the noise generated by Canvas Defender to have the sanitized and thus, the original Canvas fingerprint of the user.

2 Canvas Fingerprinting

Definition 1. The **browser fingerprinting**, refers to the unique identification of a web browser based on various attributes. This information can be collected by websites to track users and create a unique identifier for their browser. With all the information collected. The probability of two different users to have the same fingerprints is very low. So these are used as unique identifier.

Definition 2. The **Canvas fingerprint** is very useful to identify users. This type of fingerprint is based on the computer architecture of the user, client of the website. The server make the user compute in a Canvas some geometrical forms. When the user compute this, it uses his CPU and his GPU and depending of them, the generated canvas is a little bit different. This result can be use as a fingerprint.

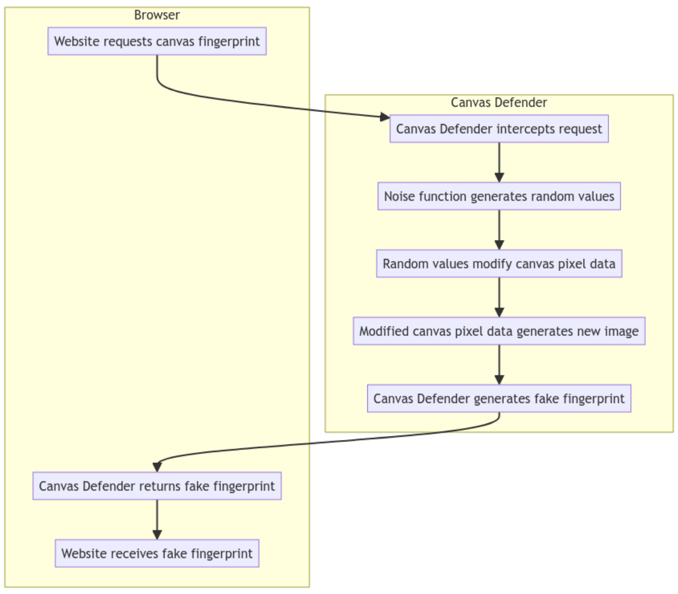
Browser
Website requests canvas fingerprint
Canvas Defender
Canvas Defender intercepts request
Noise function generates random values
Random values modify canvas pixel data
Modified canvas pixel data generates new image
Canvas Defender generates fake fingerprint
Canvas Defender returns fake fingerprint
Website receives fake fingerprint

Figure 1: canvas defender functioning

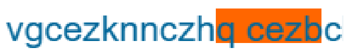

Figure 2: Canvas Image

3 Canvas Defender

Canvas defender is an extension designed to add noise the canvas fingerprinter it does this by interfering with the fingerprinting itself. The newly generated fingerprint is persistent across sessions (though you can change this behavior). An overview of the functioning can be seen in Figure 1.

Definition 3. toDataUrl is JavaScript function that generates URL-safe data from an image (like a canvas), it is encoded in base64 to be easily transferred over HTTP.

The paper [VLRR18] give a first approach as to how canvas defender implements what it does. To protect a fingerprint, Canvas Defender overwrites a Canvas object method to instead:

3.1 Overriding toDataUrl

Inside Canvas Defender source code, we can see that they override two core Javascript functions toDataUrl and toBlob (though we will only focus on toDataUrl here), they override those ones specifically because they are the only functions natively (available on all navigator's engines) capable of converting an image to base64 data.

4 Our first approach to a countermeasure

4.1 Environment

The countermeasure attack target canvas defender running on Firefox browser. To execute our test we must host a webserver permitting to the Canvas Defender addon to works, we choose to bind a simple python HTTP server serving a simple html5 page, and a JS script.

4.2 Gathering Canvas Fingerprint

When the user connects to our **Python3** server. The *index.html* page shown executes JavaScript code on the client side displaying a canvas like in figure 2.

The code executed to compute a *Canvas* object writes some text and rectangle like the studied paper [VLRR18]

```
var canvas = document.getElementById('draw');
var ctx = canvas.getContext('2d');
canvas.width = 200;
canvas.height = 50;
ctx.textBaseline = "alphabetic";
ctx.fillStyle = "#f60";
ctx.fillRect(125, 1, 62, 20);
ctx.fillStyle = "#069";
ctx.font = "16pt Arial";
ctx.fillText("vgcezknnczhq cezbchezak", 2, 20);
```

Listing 1: Canvas generator

The above code produced the followed canvas (2). Then we send the produced image to the server exporting it with the toDataurl function to be treated on the server. On the server side, we use the python PIL library to analyse the image.

4.3 Leak noise

By searching in the source code on the Canvas Defender github[1], we found out the function that generated the noise. Actually, the repository is not up to date, but it was quite interesting to analyse it and make some correlation with the paper [VLRR18].

```
function generateNewFingerPrint() {
    return new Promise((success, fail)=>{
        data = {};
        data.r = HashLength - randomIntFromInterval(0, HashLength + 10);
        data.g = HashLength - randomIntFromInterval(0, HashLength + 10);
        data.b = HashLength - randomIntFromInterval(0, HashLength + 10);
        data.a = HashLength - randomIntFromInterval(0, HashLength + 10);
        const jsonData = JSON.stringify(data);
        g_latestUpdate = Date.now();
        storageSet({"data": jsonData, "latestUpdate": g_latestUpdate}, ()=>{
            success(md5(jsonData).substring(0, HashLength));
        });
    })

}
```

Listing 2: Canvas Defender function Adding Noise

The paper showed code to add on the server side *index.html* to detect when the toDataURL method is overwritten, and moreover, detect the noise generated.

```
var o = new MutationObserver((ms) => {
    ms.forEach((m) => {
      var script = "overrideDefaultMethods";
      if (m.addedNodes[0] && m.addedNodes[0].text) {
        if (m.addedNodes[0].text.indexOf(script) >
          -1) {
          console.log("Found noise");
          var noise = m.addedNodes[0].text.match
            (/\d{1,2},\d{1,2},\d{1,2},\d{1,2}/)
          [0].split(" ,");
          console.log(noise);
        }

      }
    });
  });
  o.observe(document.documentElement, {
    childList: true, subtree: true
  });
```

Listing 3: Noise Value Detection

With this code, we create a MutationObserver that will watch for changes in the DOM and start the associated callback to function to catch the noise added by canvas defender, if we detect this kind of behavior then we know that canvas defender is active. With that done in our setup once the noise is detected we send it to our *upload* endpoint.

[1]https://github.com/jomo/canvas-defender-firefox/tree/master/js

4.4 Canvas Fingerprint Sanitization

When receiving a noised fingerprint, We convert the image in a PIL object, and subtract each pixel with the noise vector reveled.

```python
def fingerprint_from_noised(noised: str, r: int, g: int, b: int, a: int):
    noised = noised.split(',')[1]
    noised_canvas = Image.open(BytesIO(base64.b64decode(noised)))
    pix = noised_canvas.load()
    for i in range(noised_canvas.size[0]):
        for j in range(noised_canvas.size[1]):
            pr, pg, pb, pa = pix[i,j]
            pix[i,j] = (pr - r, pg - g, pb - b, pa - a)
    noised_canvas.save("unnoised.png")
    buffered = BytesIO()
    noised_canvas.save(buffered, format="png")
    img_str = base64.b64encode(buffered.getvalue())
    return img_str
```

Listing 4: Canvas Sanitation Code

The first result we obtain here is that the sanitation don't permit to obtain the raw fingerprint.

4.5 Measurement

This section describes the manners to get data and how to analyse them.

1. Obtain the original raw canvas fingerprint of the user (by using the toDataUrl), we store the obtain image on the server.
2. On client side, activate the addon to add noise, refresh the webpage to get the new canvas fingerprint. In this step we also send the noise by leaking it (describe in the Leak noise section)
3. At this step, on server side, we got two images: the raw and the noised. Revert the noise by subtracted the added noise.
4. Compare the unnoised image with the raw image
5. Redo at from the step 2 with a new noise

With this function, we can now, in theory, sanitize the noised canvas. But we don't

4.6 Why can't we get the original fingerprint

For now, we can gather the canvas fingerprint of a user. We can also detect when the user is tricking us with the Canvas Defender plug-in in his browser to noise his canvas. What we want now is to reverse the noised canvas and gather the sanitized Canvas fingerprints. For a same user, we want our server to have the same Canvas fingerprint whether the user is using Canvas Defender or not.

5 The winning countermeasure

After struggling quite a bit with the first method we decided to radically change methods, as we already knew Canvas Defender overrides some default methods from the web browser mainly *toDataUrl* so we went about restoring the original one.

To do so we add an idea that came from earlier tests about iframes, for our purpose an iframe (short for "inline frame") is an HTML element that allows a web page to embed another HTML document within it. When an iframe is loaded, it behaves like a separate web page with its own document object model (DOM), which is independent of the parent page's DOM. And importantly the parent page and the embedded document can communicate with each other meaning we can get the method from the iframe so that it can be used to replace the overridden *toDataUrl* in the parent page. (3)

5.1 Implementation details

We first create an iframe that we don't display to the user as to not arouse suspicions,

```
const iframe = document.createElement('iframe');
iframe.style.display = 'none';
document.body.appendChild(iframe);

const originalToDataURL = iframe.contentWindow.HTMLCanvasElement.prototype.toDataURL
    ;
const originalToBlob = iframe.contentWindow.HTMLCanvasElement.prototype.toBlob;

document.body.removeChild(iframe);

HTMLCanvasElement.prototype.toDataURL = originalToDataURL;
HTMLCanvasElement.prototype.toBlob = originalToBlob;
```

Listing 5: Restore default methods

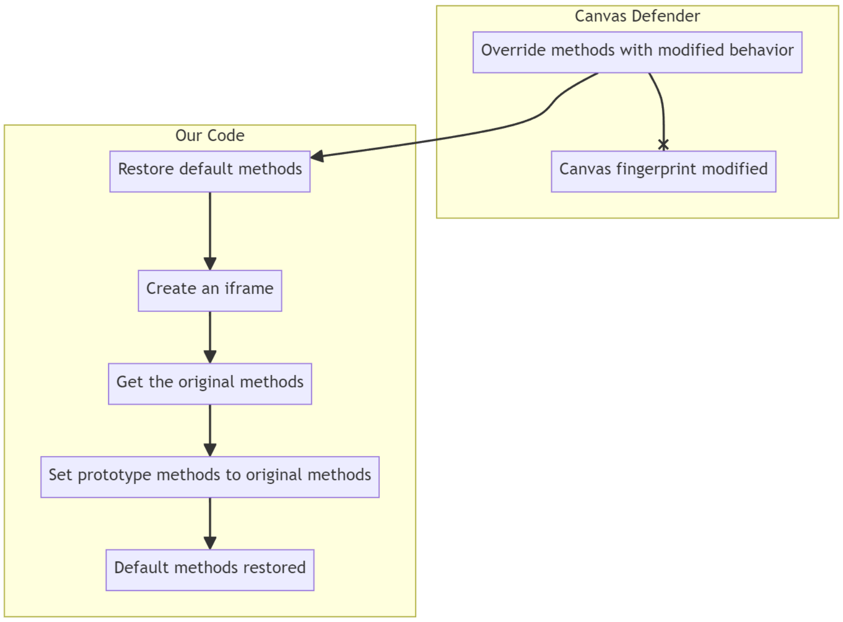

Figure 3: Diagram of the iframe attack

5.2 Potential mitigations

To mitigate this attack, it is important to ensure that any modifications to the `HTMLCanvasElement` prototype are carefully reviewed and tested to ensure that they do not introduce security vulnerabilities. Additionally, developers can use techniques such as sandboxing and input validation to further protect canvas elements from potential attacks.

6 Conclusion

To conclude this report, we saw that in the studied paper [VLRR18] the attack proposed to reverse the noise of Canvas Defender is not efficient (maybe not up to date). Instead of trying to reverse the noise. We attacks directly Canvas Defender to make it unable to execute on the canvas we gave. This one was pretty efficient and we broke Canvas Defender.

References

[VLRR18] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In *Pro-*

ceedings of the 27th USENIX Security Symposium, Baltimore, United States, August 2018.