Canvas Deceiver - A New Defense Mechanism Against Canvas Fingerprinting

Muath A. OBIDAT
Center for Cybercrime Studies, City University of New York
New York, NY 10019

Suhaib OBEIDAT
Computer Science, Bloomfield College
Bloomfield, NJ, 07003

Jennifer HOLST
Center for Cybercrime Studies, City University of New York
New York, NY 10019

Taeho LEE
Department of Computer Science, John Jay College of Criminal Justice
New York, NY 10019. USA

ABSTRACT

Browser fingerprinting refers to a collection of techniques used to gather information about a user's browser attributes. The information gained from a browser fingerprint can be used to partially or fully identify a user without using any other technique, e.g., cookies. One type of browser fingerprinting is canvas fingerprinting which utilizes HTML-canvas elements to identify users. Various defense algorithms against canvas fingerprinting have been developed, but unfortunately, have been shown to be penetrable and detectable.

In this paper, we present Canvas Deceiver, a new countermeasure against canvas fingerprint. Canvas Deceiver is a browser extension that uses a new algorithm that is different from existing problem-possessing algorithms. Canvas Deceiver does not rely on randomness, does not provide a unique identity, and is not detectable. To show its functionality and effectiveness, we tested Canvas Deceiver using different tools that provide browser fingerprint tests. According to the test results, Canvas Deceiver outperforms current countermeasures in detectability while providing sufficient anonymity to its users. For instance, in Browserleaks, the user originally was put into a group with 634 people. After using Canvas Deceiver, he is put into a group with 7847 people.

Keywords: Canvas Deceiver, Canvas Fingerprinting, JavaScript, Browser Extension, Browser Fingerprinting, Privacy

I would like to express my gratitude to Professor Abdullah Al Hayajneh for his thorough peer-editing of this paper. Professional Security Studies at New Jersey City University

1. INTRODUCTION

Ever since the inception of the Internet of Things (IoT), information protection has been a hot area of IoT research. Users' personal information, e.g., shopping behavior and items of interest, has great value now, hence an increase in user tracking activities [28]. In addition, the method used for user tracking has become significantly more sophisticated. Previously, users were tracked using stateful information such as cookies. Now, websites do not rely on stored information, rather, they use stateless information captured through browser fingerprinting [24, 25].

In the early days of the World Wide Web (WWW), each browsers rendered web content differently. To provide a better experience for users, the user-agent header was created and provided information about the user browser and system, allowing a website to customize its presentation [19]. The advent of JavaScript further allowed websites to dynamically present content; recent statistics show that around 97% of websites use JavaScript [27]. Cookies, small files stored locally on the user's computer that aid in user tracking by websites, are visible to users and can be cleared. As browsers have given more control over third-party cookies to users, interest has grown in browser fingerprinting as a replacement [18, 21]. Browser fingerprinting is not visible to users unless specifically identified as being used [15].

While browser fingerprinting has been seen negatively in terms of enabling user tracking, it has also been used in more positive ways, including bot detection, fraud detection, authentication, and improving website usability

and operation [3, 15, 19]. This more positive use of browser fingerprinting endeavors to identify a profile of a good user, while attackers try to evade detection by either posing as a legitimate user or avoiding appearing to be a known bad user [3]. Nevertheless, browser fingerprinting remains a privacy concern. Even if websites presented a privacy policy and requested user acknowledgment of cookie use, Fietkau et al. found through the use of their tool FPMON that most websites accessed fingerprint information before this acknowledgment and consent was obtained [15]. They also found that websites with especially sensitive information used a large number of fingerprinting features. To know that a user has declined consent to tracking, their fingerprint still has to be captured and compared to those who have opted out [21]. In response to these privacy concerns, the World Wide Web Consortium (W3C) has prepared guidelines and best practices for website developers to limit the fingerprinting surface and increase anonymity of users [26].

Browser fingerprinting combines various types of fingerprinting technologies to extract attributes of a user's browsing-environment. Separate pieces of information are not sufficient to identify a person, but when combined, they can be used to either partially or fully identify a user. The first large-scale analysis done by Eckersley in 2010 showed that 83.6% of 470,161 browsers that visited his testing website were uniquely identifiable [11]. A 2018 study of fingerprinting by Gómez-Boix looked at a more general-purpose website with a larger user base and found 33.6% unique fingerprints, leading to the question of effectiveness of fingerprinting on a larger scale [17, 18].

Browser fingerprinting is composed of different types of fingerprinting technologies. Examples of these technologies include user agent, canvas (two-dimensional), WebGL (three-dimensional), Web Audio, extensions, and fonts. Canvas fingerprint is independent from the attributes gathered from other fingerprinting methods, thus can be combined with other attributes easily to produce a uniquely identifiable browser fingerprint [20]. Detection of fingerprinting is complicated by the difficulty in determining whether access to particular features is due to the regular use of the web application or to an attempt to track users [15, 19].

Different types of countermeasures have been built against canvas fingerprinting. These methods include: blocking Application Programming Interfaces (APIs), altering the canvas fingerprint by adding noise, and altering the canvas fingerprint by modifying the canvas. All of these methods have their own issues including easy detection, browser malfunction, and hash collision [9, 13, 24, 25]. Using previously conducted studies and current detection methods, we will show how each of these methods is incompetent and even dangerous for user privacy when used.

In this paper, we take a different approach from currently existing measures for blocking canvas fingerprinting. Canvas Deceiver intercepts JavaScript requests and replaces the requested JavaScript file with a modified version of it. Canvas Deceiver provides static canvas fingerprints and does not touch any other parts of HTML or JavaScript, significantly reducing its detectability. This method can disguise the user into someone else hence removing user-uniqueness. We expect this method to outperform currently used methods.

In this paper, we make three contributions:

- We study currently used countermeasures against canvas fingerprinting. By using past studies, manual tests, and results obtained from using such countermeasures, we show different types of problems caused by using current countermeasures.
- We introduce a new countermeasure against canvas fingerprinting, Canvas Deceiver. We show the mechanism behind Canvas Deceiver. We also show how Canvas Deceiver is more effective than other countermeasures.
- We evaluate Canvas Deceiver effectiveness and compare that against other countermeasures using various types of browser fingerprinting tools.

2. BACKGROUND AND MOTIVATION

In this section, we provide an overview of how canvas fingerprinting is performed and currently existing countermeasures against canvas fingerprinting. We also show the shortcomings such methods suffer from.

2.1 HTML Canvas

HTML's canvas element allows for specifying two-dimensional surface that can be used for drawing raster graphics. For example, the following HTML code creates a drawing surface of size 300 by 200 pixels:

```
<canvas id="happyFace" width="300" height="200"></canvas>
```

We typically assign an id to a canvas element, so we can use this later to populate the canvas with its graphics content.

While such graphics are raster in nature, rather than vector, they are actually generated with scripting. Using instructions expressed in JavaScript, canvas can be used for drawing on the fly, which allows for responding to user events. Canvas is powerful enough to allow for creating basic illustrations, animations, or even entire applications using solely browser capabilities with no reliance on external plugins such as Adobe Flash [22].

The canvas API provides functions for creating lines and shapes, adding text, moving objects around, and applying font styles among many others. For example, using a bit of JavaScript, not shown for the sake of brevity, something such as the following can be drawn [22]:

The API also provides methods for loading images encoded as a Data URI, a plaintext representation of images. This makes it possible to embed such an image directly in the HTML document, so the browser does not have to make a separate request to get it.

2.2 Canvas Fingerprinting

Canvas fingerprinting was first introduced by Mowery et al [20]. It is done by exploiting the canvas API used to draw graphics and animations on a web page via scripting in JavaScript. A generated canvas image is rendered differently depending on the operating system (OS) type, browser type, and GPU type. The steps for generating a canvas fingerprint are:

1. A user visits a website that uses canvas fingerprinting.
2. The website calls a JavaScript-based canvas fingerprinting script.
3. The canvas fingerprinting script generates a canvas image.
4. The canvas fingerprinting script then uses ToDataURL() method to generate a data uniform resource identifier (URI), a base64 (an encoding algorithm) string representation of the canvas image. The data URI will be different depending on the type of OS, browser, and GPU used.
5. The JavaScript file uses a hash function to compute a hash of Data URI. This hash works as the canvas fingerprint of the user.
6. The website then uses the computed canvas fingerprint and combines them with other fingerprints to generate a browser fingerprint, which can be used to identify the user.
7. The canvas fingerprint can be stored in the website's server for various purposes including statistical purposes.

Hash functions are used because the data URIs are generally very long. Computing and storing a hash of a data URI results in better resource utilization. A study conducted by Englehardt et al. shows that 1.6% of the Alexa top 1 million sites used canvas fingerprinting by year of 2016 [14]. Independently, Acar et al show that 5% of the top 100,000 websites employ it [1]. More recently, Fietkau et al. find that 17.85% of the Alexa top 10,000 sites used canvas fingerprinting [15].

The mechanism behind canvas fingerprinting is actually very simple. By using a simple JavaScript library called “Fingerprint.js” [16], any website can take a hash of data URI from any user enabled canvas API and use it as the canvas fingerprint. “Fingerprint.js” does all the necessary work of creating a canvas, creating a canvas image, taking a data URI using toDataURL(), and taking a hash out of it. And because it has simple operations, all JavaScript-based canvas fingerprinting scripts are similar in nature. The only significant differences between them are the way of taking the hash out of data URI and the type of canvas image that is being created.

2.3 Canvas Fingerprinting Countermeasures

Because of its simple design, there are many ways to block websites from getting canvas fingerprints or to provide them with incorrect data. One way of blocking canvas fingerprinting is to disable the canvas API entirely. However, this prevents the websites from creating any canvas or canvas images. This type of countermeasure is implemented in tools such as Canvas Blocker [6, 12]. The second way of blocking canvas fingerprinting is by altering the data URI that is extracted from the rendered canvas image. The alteration can be done to either the data URI retrieved using ToDataURL() or directly to the canvas image. Generally, tools add random noise to the canvas image so it can be rendered differently by users. This is currently the most widely used method. Canvas Blocker, Canvas Defender, and NONYM!ZER are all known to use

this method [6, 7, 13]. Randomization was also the method used more generally by Nikiforakis et al. because they asserted that tracking was not just about trying to identify unique individuals, but being able to link different website visits by the same user [21]. By making each visit look different, linkability would harder to achieve.

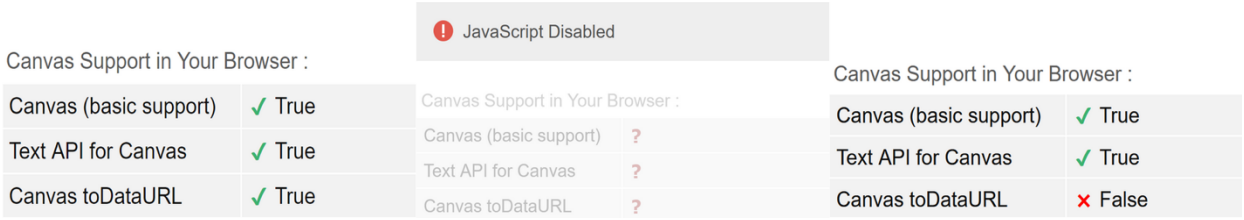
2.4 Limitations of Current Countermeasures

The aforementioned two methods possess a host of problems that can be exploited resulting in the ability to uniquely identify users. The first method, disabling the canvas API in entirety, can cause inconvenience for users as it prevents all canvas images from being drawn. Users basically lose a function of their browser, just to prevent it from being exploited as a canvas fingerprint. Another problem of disabling the canvas API is the detectability of that action. Browser fingerprinting in general is a very unfamiliar subject to the majority of Internet users. An even smaller portion of users would disable canvas API requests. For being in this small group of users who do not have a working canvas, one can be tracked by the websites as easily as any other users who have a working canvas. Not only that, websites can know whether canvas API is partially or fully blocked. Partially blocking the canvas API has the same problem. If method ToDataURL or any other parts of canvas is disabled, the website notices this and puts the user into the category of "user with canvas fingerprinting blocked". Due to these reasons, disabling the canvas API is considered an incompetent approach when compared to alternation of canvas image and addition of random noise. Figure 1 shows these problems.

The second way of blocking canvas fingerprinting is by adding randomized noise. This method detects ToDataURL requests and performs a man-in-the-middle attack on the website. It intercepts the canvas image and alters it by adding noise to it. The noise can be added to the red, green, and blue value of the canvas image or any other components of it. As a result, ToDataURL() produces a different string. In general, using randomization to make every website visit look different is difficult because impossible combinations of information may be created, and these combinations can be identified [21].

The alteration method can be static or can be volatile (new alteration is done after each refresh of the web page) each possesses its own problems If the alteration is volatile, the website can detect the fact that only the canvas fingerprint of the user is changing while other browser fingerprints such as user agent, IP address, and font are staying the same. Static alteration on the other hand suffers from hash collision. That is, the string produced before and after alteration can be the same rendering the alteration meaningless. The probability of getting hash collision depends on the randomizing function that is used for alteration. For instance, NONYM!ZER, a framework created by ElBanna et al., has a 0.3% to 1.1% rate of hash collision depending on the OS type [13].

Figure 1: Website detecting partial/full block of canvas API / JavaScript [4]. Left to right: Normal, JavaScript Disabled, and Canvas Blocker.



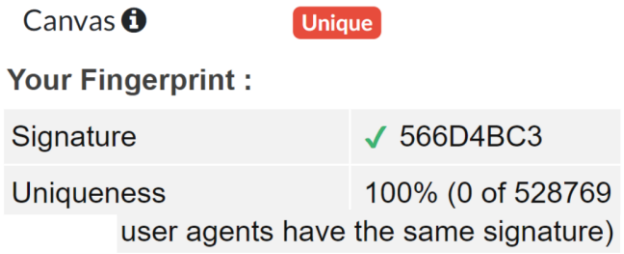


Figure 2: Canvas Blocker producing 100% unique canvas fingerprints [2, 5].

Alteration methods in general also suffer from a range of issues. One problem with this approach is distinguishability. For any user, it is nearly impossible to have a 100% unique canvas fingerprint, especially nowadays that websites have gathered hundreds of thousands, or even millions of, canvas fingerprints. Regardless of one's computer configuration, there is probably a group of people who have the same configuration that affect the canvas fingerprints. Because of that, if a person has a perfectly unique canvas fingerprint, as shown in Figure 2, that alone can be used to narrow down the user's identification. Another problem with alteration is detectability. Just like the request

Table 1: Detection of canvas alteration [10]. Top: Test result without Canvas Blocker Bottom: Result with Canvas Blocker

Fingerprint Spoofed	Original
Is browser canvas fingerprint spoofed	false
Fingerprint ID When: before DOM load Method: random image with color mixing	c86bd5907a11ad11b99740b2be7755e1b3df3bf4f2acef933ba9da93041b6eb5
Fingerprint ID When: after DOM load Method: random image with color mixing	c86bd5907a11ad11b99740b2be7755e1b3df3bf4f2acef933ba9da93041b6eb5
Is fingerprint spoofed by random noise	false

Fingerprint Spoofed	Canvas Blocker
Is browser canvas fingerprint spoofed	true
Fingerprint ID When: before DOM load Method: random image with color mixing	580ec1e645074a1604e81af6b22cec00d7b704d41361439ddc4207e0951ae4e1
Fingerprint ID When: after DOM load Method: random image with color mixing	1ab343c8e26e80429431e2cfb5658704c04dd1e3b773e45af88c147458b46fdf
Is fingerprint spoofed by random noise	true

blocking method, the alteration method can easily be detected by websites. A website can take a canvas fingerprint before Document Object Model (DOM) loads, and take the canvas fingerprint again after DOM has loaded. Then it can compare both canvas fingerprints, if the fingerprints are different, it means the canvas fingerprint is spoofed. Webbrowsertools provides a simple test that can detect canvas fingerprint spoofers [10]. Through manual tests, we found that current methods such as Canvas Blocker and Canvas Defender can be detected using this test. The result is shown in Table 1. As we can see, the fingerprint IDs change for the test result with Canvas Blocker. Without Canvas Blocker, the test gives a uniform test result. This makes these methods very vulnerable as usage of canvas fingerprint spoofers can be uniquely identifying of a user.

Yet another approach to mitigate fingerprinting in general is normalization, where an attempt is made to make all devices and users look the same; this is the approach taken by the Tor and Brave browsers [18, 19]. Researchers are also investigating machine learning approaches, including FP-Inspector from Iqbal et al., to detect and mitigate fingerprinting [18].

3. THE PROPOSED SCHEME CANVAS DECEIVER

Canvas Deceiver is the method we introduce, which overcomes the problems current canvas fingerprinting countermeasures suffer from. In this section, we discuss the mechanism behind Canvas Deceiver and compare it to existing countermeasures.

3.1 Mechanism

For Canvas Deceiver, we employ direct modification of the JavaScript file as our method. From investigation, we found that a lot of canvas fingerprinting methods use the same canvas image and the same JavaScript file for its fingerprinting. This means that neutralizing popular JavaScript files such as Fingerprint.js [16] can result in significant progress in blocking canvas fingerprinting.

The mechanism behind Canvas Deceiver is very simple. If a website makes a request to a canvas fingerprinting JavaScript file, Canvas Deceiver intercepts the request and gives a modified version of it, which is stored inside Canvas Deceiver's local library. The replaced JavaScript file acts the same way as the original version. The only difference is that it gives a persistent, predetermined string

when ToDataURL() is called. The scheme is shown in Figure 3.

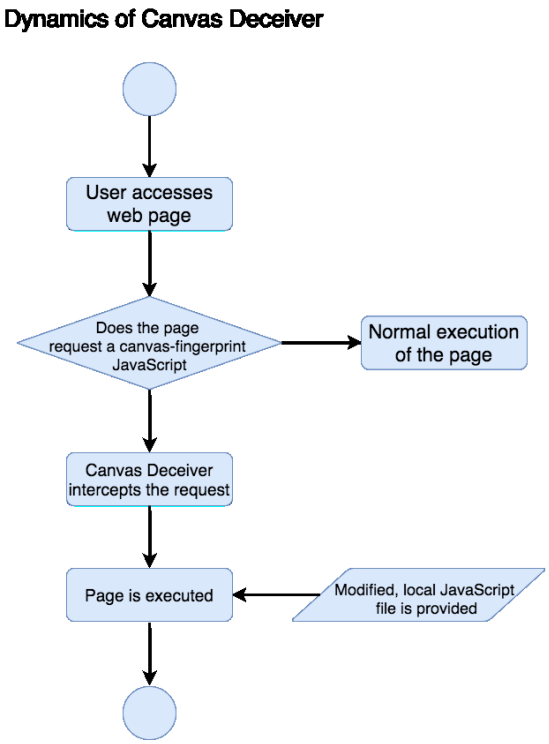


Figure 3: Dynamics of the Proposed Scheme - Canvas Deceiver

Canvas Deceiver can be implemented by two different methods. One way is by using Chrome DevTools [8]. Users can manually override parts of the content of any website. By manually overriding and making modifications to canvas fingerprinting JavaScript file, any user can implement the method that is used by Canvas Deceiver. Once the user makes override and replaces the JavaScript file with a modified, local version, Chrome browser automatically uses the locally modified JavaScript file.

Another way of implementing Canvas Deceiver method is through a browser extension. Canvas Deceiver is being developed as a browser extension with a library of canvas fingerprinting JavaScript files. Canvas Deceiver grabs the JavaScript request of a website with chrome web Request and hands in the modified version of the requested JavaScript file. We also want to further develop Canvas Deceiver to react to ToDataURL().

The local JavaScript files are currently modified manually. We retrieve widely used JavaScript files from websites and modify them to produce desired predetermined canvas fingerprints. Normally, strings are obtained from ToDataURL() and passed to a variable. We change this and pass a fixed predetermined string to the variable. This simple modification is shown in Listing 1. Here, o = c.toDataURL ("image/png") is replaced with o = 'predetermined string'. The JavaScript file then takes a hash of this predetermined string. As a result, a generic canvas fingerprint is produced by the JavaScript file. We then store them inside Canvas Deceiver's JavaScript library. When each JavaScript is being requested by the website, Canvas Deceiver intercepts the request and gives the modified version from the library to the website.

3.2 Evaluation and Results

Due to its behavior and mechanism, Canvas Deceiver overcomes most of the problems that current canvas fingerprinting countermeasures possess. Detectability is one of the most crucial and commonly found problems in current canvas fingerprinting countermeasures. Canvas Deceiver's strongest point is its invisibility. Firstly, none of the canvas API or JavaScript requests in general are blocked by Canvas Deceiver. The user gets a perfectly functioning browser. Secondly, no alteration is involved in Canvas Deceiver. Canvas Deceiver provides a uniform string, so taking canvas fingerprints multiple times has no effect on it. In addition, Canvas Deceiver does not interact with created canvas images. Investigation of canvas images also has no effects on Canvas Deceiver.

To evaluate Canvas Deceiver's detectability, we ran the same tests that we performed on other tools in section 2. As shown in Figure 4, the same test that detected random noise could not detect Canvas Deceiver. The test could not spot the fingerprint spoofing done by Canvas Deceiver. Compared to existing tools like Canvas Blocker and Canvas Defender, Canvas Deceiver has a huge advantage in detectability.

Another strength of Canvas Deceiver is its indistinguishability. The strings produced by Canvas Deceiver are predetermined data we gathered using popular canvas fingerprinting images and settings of a mass-produced computer model. Because of this, the hash taken from the provided string looks very generic. Current canvas fingerprinting countermeasures produce unique canvas fingerprints, exposing users and making them easily trackable. This is not a problem for users of Canvas Deceiver. Because of its generic string, the canvas fingerprints produced from our modified JavaScript files look completely normal. This result lets the users blend into a huge group of people who have the same canvas fingerprints. As shown in Figure 5, the uniqueness after

```
var canvasData;
try {
  var canvas = document.createElement('canvas');
  canvas.height = 60;
  canvas.width = 400; // Canvas is generated
  var canvasContext = canvas.getContext('2d');
  canvas.style.display = 'inline';
  canvasContext.textBaseline = 'alphabetic';
  canvasContext.fillStyle = '#f60';
  canvasContext.fillRect(125, 1, 62, 20);
  canvasContext.fillStyle = '#069';
  canvasContext.font = '11pt no-real-font-123';
  canvasContext.fillText("Cwm fjordbank glyphs vext quiz, \uD83D\uDE03", 2, 15);
  canvasContext.fillStyle = 'rgba(102, 204, 0, 0.7)';
  canvasContext.font = '18pt Arial';
  canvasContext.fillText("Cwm fjordbank glyphs vext quiz, \uD83D\uDE03", 4, 45); // Canvas image is generated
  canvasData = canvas.toDataURL(); // Data URI string is generated and passed to canvasData
  Canvas Deceiver Modification
  canvasData = '
  // Data URI string is replaced with a fixed string
```

Listing 1: Modification of canvas fingerprinting JavaScript file. Canvas Deceiver adds a code to replace data URI of canvas image [2].

using Canvas Deceiver drops by a significant margin. For instance, in Browserleaks, the user originally was put into a group with 634 people. After using Canvas Deceiver, he is put into a group with 7847 people. The user is now placed in a much bigger group, which makes it harder for websites to narrow down the user's identity. Similarly, the AmIUnique test had non-uniqueness increase from 0.64% to 1.6%. Using 1,720,569 as the total number of collected fingerprints, this represents that the user is put into a group with 16,517 more people. These results show that the usage of Canvas Deceiver mitigates canvas fingerprinting considerably.

Canvas Deceiver's independence from randomization is another advantage to Canvas Deceiver. By observing current canvas fingerprinting countermeasures, we found that randomization is responsible for many different problems. Randomization is responsible for: uniqueness of canvas fingerprints, detectability, noticeable difference in canvas image, and hash collisions.

Canvas Deceiver is also free from hash collisions. The strings produced by Canvas Deceiver come from one uniform computer and browser setting. This setting is very generic, so a lot of users are using the same settings. Because of that, it is very possible for a user to have the exact same canvas fingerprint as the one produced by Canvas Deceiver.

The user, however, does not need to worry about this because it means the user's canvas fingerprint is too generic to be used for browser fingerprinting. Further, as more users use Canvas Deceiver, the canvas fingerprints produced from Canvas Deceiver will become even more insignificant.

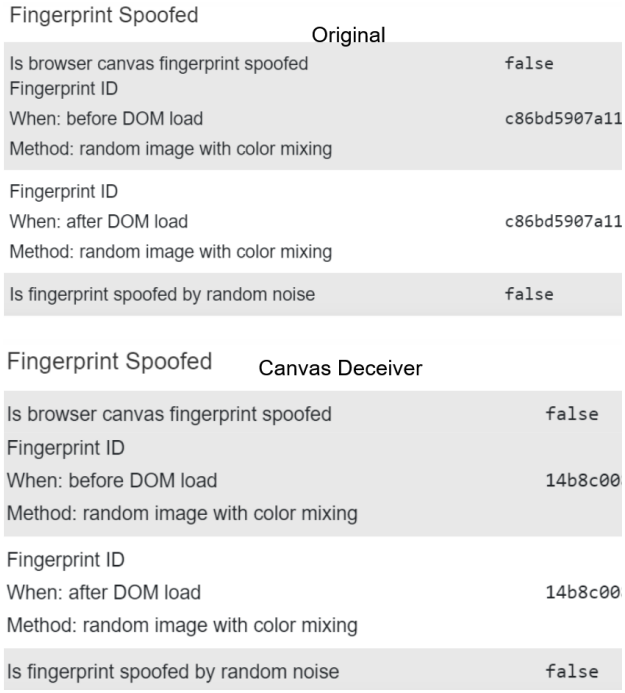


Figure 4: Detectability of Canvas Deceiver [10].

4. DISCUSSION AND FUTURE DEVELOPMENT

By using multiple browser fingerprinting sources, we tested currently used canvas fingerprinting countermeasures such as Canvas Blocker and Canvas Defender along with Canvas Deceiver. During the test, Canvas Deceiver outperformed all of its contestants. The criteria of the tests were: detectability, uniqueness, and functionality. Canvas Deceiver provided a non-distinguishable, uniform canvas fingerprint that is not

detectable. It did not harm the browser or the canvas image in any way.

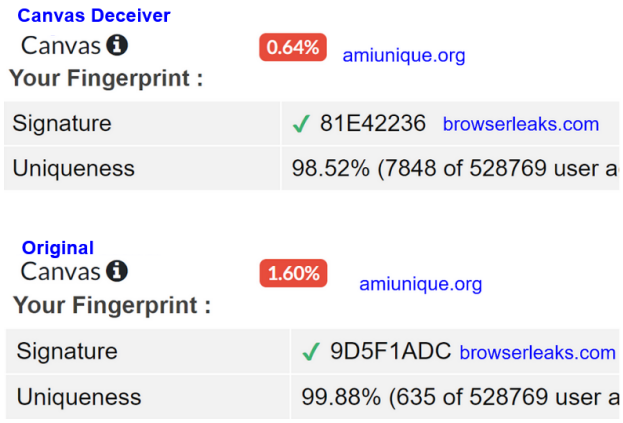


Figure 5: Outcomes of Canvas Deceiver [2, 4].

Canvas Deceiver is still in development. There are 2 major assignments for Canvas Deceiver. One is getting a bigger library. As of now, Canvas Deceiver can only respond to a handful of websites. Getting its library bigger so it can respond to many different websites is very important. We can also make Canvas Deceiver more compact by finding out different ways of intercepting JavaScript files and modifying them.

Another assignment is to determine the most generic computer setting. Takasu et al. conducted a study on this topic in 2015 [23]. We can conduct another large-scale study similar to this with more recent data retrieved from popular browser fingerprinting sources such as AmIUnique [2]. Implementing the most generic canvas fingerprint would enhance Canvas Deceiver significantly.

5. CONCLUSION

A recent crawling result showed decrease in usage of canvas fingerprinting [14], while another recent study using a different methodology showed an increase [17]. In any event, canvas fingerprinting is still a very dangerous fingerprinting technique which can be a crucial part of browser fingerprinting. Compared to the currently existing canvas fingerprint countermeasures such as Canvas Blocker, Canvas Defender, and Canvas Fingerprint Defender, Canvas Deceiver has advantages in the following criteria: (i) detectability, (ii) uniqueness, (iii) uniformity, and (iv) functionality of the browser.

We plan to extend this work by implementing Canvas Deceiver as a browser extension. We also plan on expanding the library of canvas fingerprinting JavaScript files that we make use of. To further verify the effectiveness of the proposal, we also plan on testing it against a much larger set of websites and comparing its performance to counterpart anonymizing schemes. We also plan on combining Canvas Deceiver with other effective countermeasures against other types of browser fingerprinting and explore the effectiveness of such a synergetic scheme.

6. ACKNOWLEDGMENT

Funding for this work was provided by a faculty scholarship grant from the Office for the Advancement and Research at John Jay College.

7. REFERENCES

[1] G. Acar, C. Eubank, S. Englehardt, M.Juarez, A. Narayanan, and C. Diaz. “The Web Never Forgets,” **Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS 14**, 2014.

[2] AmIUnique. Accessed on: May 17, 2020. [Online]. Available at https://amiunique.org/

[3] B. A. Azad, O. Starov, P. Laperdrix, and N. Nikiforakis. “Short Paper-Taming the Shape Shifter: Detecting Anti-fingerprinting Browsers,” **International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment**, June 2020, Springer, Cham., pp. 160-170.

[4] V. Bernardo, and D. Domingos. “Web-Based Fingerprinting Techniques,” **Proceedings of the 13th International Joint Conference on e-Business and Telecommunications** – Vol. 4: SECRYPT, (ICETE 2016), pp. 271-282.

[5] Browser Leaks. Accessed on: May 17, 2020. [Online]. Available at: https://browserleaks.com/canvas

[6] CanvasBlocker. Accessed on: May 17, 2020. [Online]. Available at: https://github.com/kkapsner/CanvasBlocker

[7] Canvas Defender. Accessed on: May 17, 2020. [Online]. Available at: https://multilogin.com/canvas-defender/

[8] Chrome DevTools. Accessed on: May 17, 2020. [Online]. Available at: https://developers.google.com/web/tools/chrome-devtools

[9] A. Datta, J. Lu, and M. C. Tschantz. “Evaluating Anti-Fingerprinting Privacy Enhancing Technologies,” **The World Wide Web Conference 2019**. https://doi.org/10.1145/3308558.3313703.

[10] Detecting Canvas Fingerprint Spoofer. Accessed on: May 17, 2020. [Online]. Available at: https://webbrowsertools.com/canvas-fingerprint/

[11] P. Eckersley, "How unique is your web browser?" **International Symposium on Privacy Enhancing Technologies Symposium**, 2010, Springer, pp. 1–18.

[12] A. ElBanna and N. Abdelbaki, "Browsers Fingerprinting Motives, Methods, and Countermeasures," **2018 International Conference on Computer, Information and Telecommunication Systems (CITS)**, Colmar, 2018, pp. 1-5.

[13] A. ElBanna and N. Abdelbaki, "NONYM!ZER: Mitigation Framework for Browser Fingerprinting," **2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)**, Sofia, Bulgaria, 2019, pp. 158-1.

[14] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," **Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security**, 2016, pp. 1388–1401.

[15] J. Fietkau, "The Elephant in the Background: A Quantitative Approach to Empower Users Against Web Browser Fingerprinting," No. 4473, EasyChair, 2020.

[16] Fingerprint.js. Accessed on: May 17, 2020. [Online]. Available at: https://github.com/Valve/fingerprintjs2/blob/master/fingerprint2.js

[17] A. Gómez-Boix, P. Laperdrix, B. Baudry, "Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale," **WWW2018 - TheWebConf 2018: 27th International World Wide Web Conference**, Apr 2018, Lyon, France. pp.1-10, ⟨10.1145/3178876.3186097⟩, ⟨hal-01718234v2⟩

[18] U. Iqbal, S. Englehardt, and Z. Shafiq, "Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors", 2020, arXiv preprint arXiv:2008.04480.

[19] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine, "Browser fingerprinting: a survey," **ACM Transactions on the Web (TWEB)**, 14(2), 2020, pp. 1-33.

[20] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5", **IEEE Web 2.0 Workshop on Security and Privacy (W2SP)**, 2012.

[21] N. Nikiforakis, W. Joosen, and B. Livshits, "Privaricator: Deceiving fingerprinters with little white lies," **Proceedings of the 24th International Conference on World Wide Web**, May 2015, pp. 820-830.

[22] J. N. Robbins, "Embedded Media" in **Learning Web Design, 5th Edition**, O'Reilly Media, 2018, Chapter 6, pp. 228 – 232.

[23] K. Takasu, T. Saito, T. Yamada and T. Ishikawa, "A Survey of Hardware Features in Modern Browsers: 2015 Edition," **International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing**, 2015, pp. 520-524. doi: 10.1109/IMIS.2015.72

[24] A. Vastel, P. Laperdrix, W. Rudametkin and R. Rouvoy, "FP-STALKER: Tracking Browser Fingerprint Evolutions," **2018 IEEE Symposium on Security and Privacy (SP)**, San Francisco, CA, 2018, pp. 728-741.

[25] A. Vastel, P. Laperdrix, W. Rudametkin and R. Rouvoy, "FP-Scanner: The privacy implications of browser fingerprint inconsistencies," **27th USENIX Security Symposium (USENIX Security 18)** (Baltimore, MD), USENIX Association, August 2018, pp. 135–150.

[26] W3C, "Mitigating Browser Fingerprinting in Web Specifications," W3C Interest Group Note 28, March 2019. Accessed on November 27, 2020. [Online]. Available at: https://www.w3.org/TR/fingerprinting-guidance/

[27] W3Techs, "Usage statistics of JavaScript as client-side programming language on websites," Accessed on November 27, 2020. [Online]. Available at: https://w3techs.com/technologies/details/cp-javascript.

[28] Z. Yu, S. MacBeth, K. Modi, J. M. Andpujol, "Tracking the trackers," In **Proceedings of the 25th International Conference on World Wide Web (2016)**, International World Wide Web Conferences Steering Committee, pp. 121–132.