Morellian Analysis for Browsers: Making Web Authentication Stronger With Canvas Fingerprinting

Pierre Laperdrix[1], Gildas Avoine[2], Benoit Baudry[3], and Nick Nikiforakis[4]

[1] CISPA Helmholtz Center for Information Security
[2] INSA, IRISA, IUF
[3] KTH
[4] Stony Brook University

Abstract. In this paper, we present the first fingerprinting-based authentication scheme that is not vulnerable to trivial replay attacks. Our proposed canvas-based fingerprinting technique utilizes one key characteristic: it is parameterized by a challenge, generated on the server side. We perform an in-depth analysis of all parameters that can be used to generate canvas challenges, and we show that it is possible to generate unique, unpredictable, and highly diverse canvas-generated images each time a user logs onto a service. With the analysis of images collected from more than 1.1 million devices in a real-world large-scale experiment, we evaluate our proposed scheme against a large set of attack scenarios and conclude that canvas fingerprinting is a suitable mechanism for stronger authentication on the web.

1 Introduction

Passwords are the default solution when it comes to authentication due to their apparent simplicity for both users and developers, as well as their ease of revocation [11]. At the same time, there is a constant stream of data-breach-related news where massive lists of credentials are exfiltrated from high-profile companies [3, 5]. These credentials are subsequently weaponized against other websites via automated bots which brute-force login forms, abusing the phenomenon of password reuse.

Even though many services support 2-Factor-Authentication (2FA) which largely addresses the issue of credential theft [6], the adoption of 2FA is still low. According to a 2017 survey [15], only 28% of users utilize 2FA (not necessarily across all of their accounts) with more than half of the respondents not knowing what 2FA is. A talk of a Google engineer in January 2018 also revealed that less than 10% of active Gmail users have activated 2FA [22].

In this environment, we argue that there is a need for a technical solution that bridges the gap between the insufficiency of passwords and the low onboarding of 2FA. To that extent, we propose a novel browser-fingerprinting-based authentication protocol which relies on canvas fingerprinting for differentiating between a user's real browser and that of an attacker who is impersonating the user. We call this a *Morellian Analysis* of browsers, referring to Giovanni Morelli's techniques for distinguishing real paintings from forgeries by relying on idiosyncrasies and repeated stylistic details of artists [28].

Browser fingerprinting has received significant attention in the last few years with many researchers studying the adoption of browser fingerprinting in real websites [9, 8, 17] and proposing new types of fingerprinting techniques [35, 23, 32] as well as defenses against fingerprinting [25, 21, 19]. Even though the majority of research has focused on the negative aspects of browser fingerprinting, some research has discussed its usage for authentication purposes [10]. Unfortunately, as noted by Spooren et al. [33], the main challenge with using browser fingerprinting for authentication is that most collected attributes of a browser are static (i.e., they are constant and do not depend on any input) and therefore can easily be modified and replayed, enabling attackers to impersonate users. For example, an attacker who can lure users to phishing pages can also collect their browser fingerprints and replay them to the benign service at a later time, when trying to log in with the stolen credentials.

Key insight: The key insight of our work is that active fingerprinting techniques like canvas FP [8, 20], WebGL FP[13], Audio FP[17] or Crypto FP[29] can be used to construct a challenge/response-based authentication protocol. In contrast with most browser fingerprinting techniques that query an API and collect simple and predictable browser-populated values, these techniques are dynamic. Here, we look specifically at canvas fingerprinting but the same approach can be applied to other active techniques. In our proposed authentication protocol, every time the user logs in, the browser is asked to paint a canvas element in a very specific way following a newly received challenge. Since a canvas rendering is computed dynamically through a process that relies on both the software and hardware of the device, it is much harder to spoof than static attributes, making it a strong candidate for authentication.

The authentication protocol: Our proposed authentication protocol verifies the identity of a device through canvas fingerprinting, acting as a post-password authentication scheme that is completely transparent to users. Through the process of challenge and response for each login, the user's device renders two images: one that is used to verify the current connection and the other that will be used to verify the next connection. Because canvas fingerprinting is a stable and deterministic process, the server can expect that the currently received response will match with pixel-precision the response generated during the previous connection. If any irregularities are detected, the canvas test fails while our proposed system can account for users owning multiple devices generating distinct fingerprints.

Evaluation and implementation: The dynamic nature of canvas fingerprinting is made evident by the fact that no two challenges of our proposed protocol are identical. By modifying a precise set of parameters, we are able to offer a large variety of canvas challenges that increase the complexity of our test. With the analysis of images generated by more than 1.1 million browsers, we show that our mechanism is largely supported by modern devices, since 99.9% of collected responses were real canvas renderings. We demonstrate that our protocol can thwart known attacks on both desktops and mobile devices, and describe how the knowledge of a device's complete fingerprint, does not ultimately defeat our mechanism. Finally, we quantify the overhead of our canvas-based authentication protocol and show that it is negligible and can be straightforwardly integrated with existing authentication systems.

2 Tuning canvas fingerprinting for authentication

To be suitable for authentication, canvas fingerprinting must fulfill the two following properties: i) it must be possible to generate challenges that are rendered differently on different devices (canvas tests exhibit diversity between devices) and, ii) the rendering of canvas images is stable over time (the identity of a device does not change over time). In this section, we present the pilot experiment that we conducted to establish that canvas fingerprinting can satisfy those properties.

2.1 Tweaking the right parameters

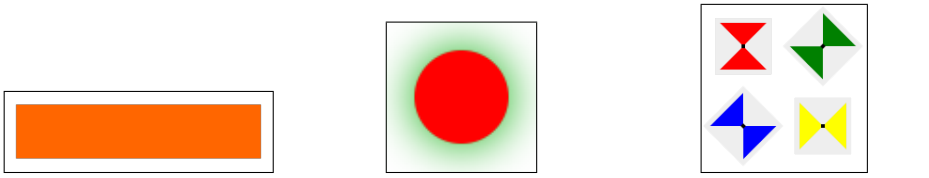
The canvas HTML element acts a drawing board that offers powerful tools to render complex images in the browser. We use the capabilities and the features of the Canvas API to their full potential to expose as much diversity as possible between devices (i.e. what types of renderings depend the most on the device's configuration).

We perform this study in three separate phases. The goal of the first phase is to determine the effectiveness of the different drawing methods available in the canvas API for identification purposes. By drawing a wide range of shapes and string with various effects, we are able to identify the type of drawing that has the most impact on the diversity of renderings. Then, with each subsequent phase, we refine the most promising methods to obtain the ones exposing the most diversity between devices. This study was performed on AmIUnique.org that allows users to inspect their browser fingerprints between January 2016 and January 2017. The participants were fully aware of the experiment and had to click on a button so that the tests were performed in their browser. The highlights of this study can be found below (additional results available here).

Entropy. In order to analyze the effectiveness of each of our tests, we use Shannon's entropy [31]. More specifically, we look at how unique each canvas rendering is to identify the drawing primitives that we can leverage for our authentication mechanism. The entropy we compute is in bits where one bit reduces by half the probability of a variable taking a specific value. In our case, the higher the entropy is, the better it is since it means that more diversity is exhibited between devices.

Tests with low entropy. Drawing simple shapes like the ones in Figure 1 does not provide enough ground for distinguishing devices. Straight lines with uniform colors provide almost identical results across the tested browsers. Curved forms helps increasing the overall entropy of a test since browsers must apply anti-aliasing algorithms to soften the edges of ellipses but, overall, the diversity of renderings is simply far too low. It does not provide a solid ground on which to build a robust authentication scheme.

Tests with high entropy. In order to benefit from a more robust solution, we have to turn to the features of the canvas API that rely a lot more on the underlying software and hardware stack. Figure 2 presents an example of a test we ran during our experiment that had a high level of entropy. After running three distinct phases, we identified the following features that contribute the most to exposing device diversity.
• **Strings.** While rendering an image, a script can request any arbitrary font. If the system does not have it, it will use what is called a fallback font. Depending on the



(a) One rectangle (Entropy: 2.32 bits) *(b) Two ellipses (Entropy: 4.15 bits)* *(c) Four squares (Entropy: 4.66 bits)*

Fig. 1: *Examples of tests with low entropy (Maximum possible entropy 13.87 bits)*

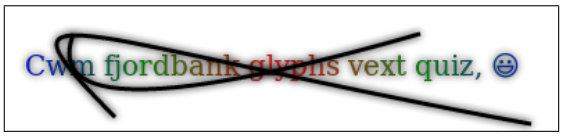


Fig. 2: *Example of a test with high entropy (Entropy: 8.11 bits)*

operating system, this fallback font differs between devices, increasing the pool of distinct canvas renderings. Because of this mechanism, strings become an essential basis on which to build an authentication mechanism as they are a strong way to exploit the natural diversity of fonts installed on a system. Moreover, as noted by Laperdrix et al., adding emojis in strings helps distinguishing devices because their actual representation differs between systems [20]. In our series of tests, all strings rendered with an emoji have a higher entropy than those without.

• **Gradients.** Having a color gradient on a string accentuates the differences observed between devices. The natural transition of colors is not rendered identically across devices. Some renderings present a very smooth transition while others have more abrupt changes where the limit between colors is clearly visible.

• **Mathematical curves and shadows.** Mathematical curves and shadows are a welcome addition to increase the entropy even further. The first depends on the math library used by the browser to position each pixel at the right location while the other relies on the graphics driver to render a shadowy cloud of pixels.

• **Combining all features.** One result we obtained is that combining different features together generate more graphical artifacts than just rendering them separately. This proves that rendering many different objects at the same location on a canvas image is not a complete deterministic process. Each device has its own way of assigning a specific value to a pixel when dealing with many different effects.

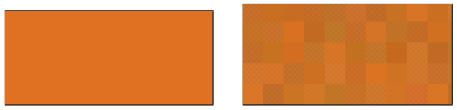
• **Size.** Performing the same canvas test on a larger surface leads to a higher entropy. Larger-sized objects increase the differences between two renderings since the edges of the different objects and the exact position of each pixel are defined more precisely.

2.2 Understanding canvas stability

In parallel with our study on canvas diversity, we performed a study on the stability of canvas renderings. Indeed, if a device produces many different results in the span of a day or a week for the exact same set of instructions, our system must take that behavior into account when asserting the validity of a response.

In October 2015, we released a browser extension on both Chrome and Firefox to study the evolution of a canvas rendering in a browser. At the time of writing, the extension is installed on more than 600 Firefox browsers and 1,700 Chrome browsers. 158 devices have been using the extension for more than a year, giving us insights about stability, which have never been discussed before. When the extension is installed, it generates a unique ID. Every 4 hours, the exact same canvas test is executed by the extension and the results are sent to our server along with the unique ID and additional system information like the user-agent or the device's platform. This way, we can follow any evolution of a canvas rendering on the same device and understand the changes. If after several months, the number of canvas changes is small, this would mean that the canvas API is suitable for our authentication mechanism.

• **Outliers.** When we started analyzing the data, we noticed some unusually high number of changes. Several devices reported different renderings every day and even one device reported 529 changes in less than 60 days. Due to the nature of the research subject, the extension attracts a biased population interested in fingerprinting. After investigating these high numbers of changes, we found out that some users deliberately modified their browser's configuration to produce variations in their own canvas renderings. Others have installed canvas poisoners that add a unique and persistent noise to the generated images. Color values of pixels are changed to produce a unique rendering every time the test is run. Poisoning a canvas element can be done via a browser extension like the Canvas Defender extension [4] or it can directly be a built-in feature. Pale Moon is the first browser to include such a feature in 2015 [1]. Figure 3 illustrates the modifications done by a canvas poisoner.



(a) Without a poisoner *(b) With a poisoner*

Fig. 3: *Impact of a canvas poisoner on a rendering*

On Figure 3a without the canvas poisoner, the surface of the orange background is uniform and all pixels have the exact same color values. On Figure 3b, the canvas poisoner modifies the RGB values and the Alpha channel of the defined pixels and it creates a unique rendering at every execution.

Because of these non-organic changes produced by these different mechanisms, we consider a device as an outlier if more than 8 canvas changes were observed during the given time period. These represent less than 4% of our dataset.

• **Results.** Figure 4 displays the main results of this study. Each boxplot represents the time for which we observed certain devices (difference between the first and the last fingerprint recorded for a device). This way, we can include every device that participated in our experiment even for a really short period of time. The mean and standard deviation values for each category were computed without taking the outliers into account.

The first observation is that in the span of several weeks and even months, the mean number of canvas changes is really low. It is around 2 for the first six months and, as time goes by, it is slightly increasing to be above 3 for the longest periods of our dataset. Even for the 158 devices that used the extension for more than a year,

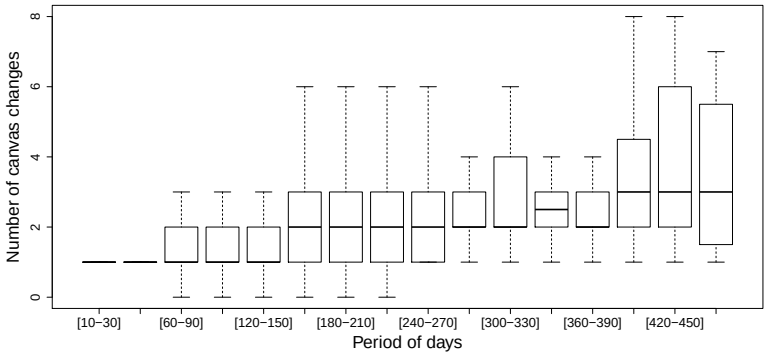


Fig. 4: *Boxplot on the number of canvas changes*

half of them present 3 or less canvas changes. This means that a canvas rendering can be even more stable than the browser's user-agent which changes every 6 weeks when a new browser version is released. Moreover, the really small values for the standard deviation also prove that the majority of canvas changes are taking place in a small range between 1 and 5. This shows that canvas fingerprinting is deterministic enough to be usable as a means of verification that can support an authentication mechanism.

The second observation is that it is uncommon to find a device where the canvas rendering has not changed for a period of several months. This can be explained by organic changes of the browsing environment. For example, we noticed that some devices on Windows had an automatic canvas change when they switched from Firefox 49 to 50 because Mozilla added built-in support of emojis directly in Firefox with the bundled EmojiOne font[2]. These changes are organic in the sense that they are caused by a "natural" update of the system and its components (e.g. a browser update or a change of graphics driver). In order to deal with these changes, our canvas mechanism relies on an additional verification from the authentication scheme to confirm the device identity as described in Section 3.2.

3 Canvas authentication mechanism

In this section, we present the core of our authentication mechanism and we describe the protocol associated with it. We also detail where it can fit within an already-existing multi-factor authentication scheme to augment it and reinforce its security.

3.1 Challenge-response protocol

The aim of this protocol is to define a series of exchanges so that a client (prover) can authenticate himself to the server (verifier). Our core mechanism relies on the comparison of images generated through the canvas browser API. During one connection to the server, the client generates a very specific image in the browser following a unique set of instructions. Our mechanism will then verify in the next connection that the device can generate the exact same image.

For our comparison, we only look to see if two generated images are identical to one another. Indeed, if two images differ even by a very low number of pixels, it could legitimately be caused by an update of the client's device or it could have been generated by a completely different device altogether. For this specific reason and to prevent making false assumptions by incorrectly identifying a device, we do not compute any similarity score between two images and we do not use specific thresholds. Moreover, as we showed in Section 2.2, generated images are sufficiently stable over time so we can perform pixel-precise comparisons without worrying about constant changes. For devices using a fingerprinting protection like a canvas poisoner, we expect users to whitelist websites that implement our canvas mechanism to avoid changes at every connection. The challenge-response protocol underlying our authentication mechanism is depicted in Figure 5.

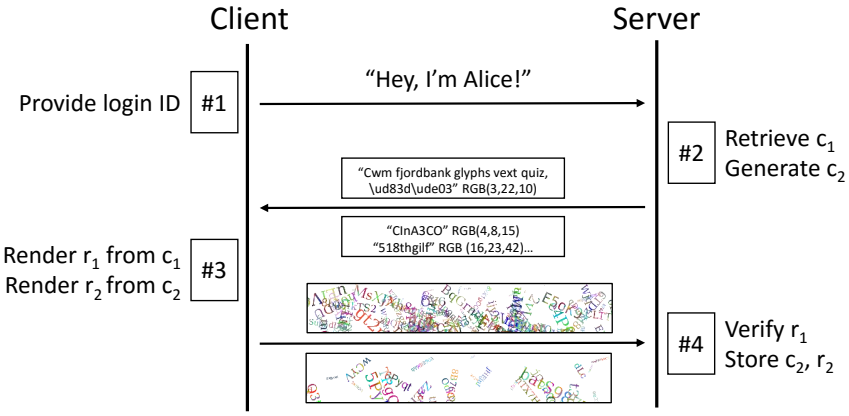


Fig. 5: *Overview of the authentication protocol*

• **Step #1.** The client sends to the server the ID associated with the user.
• **Step #2.** Based on the received ID, the server sends two canvas challenges c_1 and c_2 (i.e. two sets of instructions) to the client:
  – the challenge c_1 is one that has already been answered by the client in the previous connection. It is used to authenticate the device for the current connection by comparing the response to the one given in the last authentication process.
  – the challenge c_2 is randomly generated and will be used to authenticate the device for the next connection.

The server is trusted to generate challenges in a way that they are indistinguishable from random values.
• **Step #3.** The client executes the two sets of instructions in the user's browser and generates two canvas images. Each image is then sent to the server in a textual representation obtained with the "getDataURL()" function of the canvas API.
• **Step #4.** This is the step where the server confirms the identity of the client. The server asserts whether the given response to the first challenge matches the one of the previous connection. If the given canvas rendering is identical to the stored one (i.e. if the obtained strings are identical), we say that the response is valid and the authentication succeeds. The response r_2 is then stored alongside the associated

challenge c_2 as they will be used to authenticate the device during the next connection. If the given canvas does not match the stored one, the authentication system will follow through by asking for an additional confirmation through another channel (see 3.2).
• **Assumptions on the channel.** A confidential channel (e.g., based on HTTPS) must be used between the server and the client to avoid a trivial replay attack of a genuine authentication exchange. Moreover, there is a bootstrap phase for our mechanism because the verification process cannot be performed during the very first connection to our system where only a single challenge is sent. We assume that servers and clients share an authenticated channel in the very first connection.

3.2 Integration in a MFA scheme

As seen in Section 2.2, a single canvas rendering from the same computer evolves through time and it can take weeks or months before a difference can be observed. Because of these changes, our system cannot stand on its own and substitute a complete MFA scheme. Here, the main goal of our canvas mechanism is to strengthen the security provided by a multi-factor authentication scheme and works alongside it. It can be integrated in traditional multi-layered schemes like many 2FA ones or it can be used as an additional verification technique for authentication protocols like OpenID Connect or Facebook Login that provides end user authentication within an OAuth 2.0 framework [7]. Figure 6 gives an overview of how our system can fit within an existing MFA scheme.

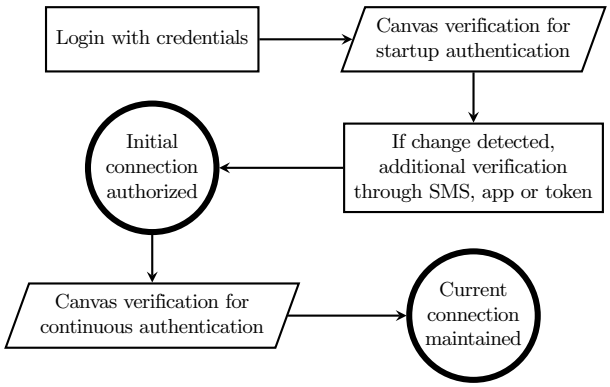


Fig. 6: *Overview of the integration of the canvas mechanism in a multi-factor authentication scheme*

• **Login verification.** First, the user must provide his credentials. For many websites, these represent the combination of a username, a password and the presence of a cookie in the browser. After this phase is successfully completed, our canvas mechanism takes over to perform the necessary verification. If the responses from the canvas tests are valid, the user is authorized to continue. If the information sent by the browser is labeled as incorrect, an additional means of verification will kick in like the use of a one-time password (OTP) via an SMS or an app. As shown in our stability study, it can take many weeks for a single canvas rendering to undergo a

change due to a "natural" evolution of the user's device. The expected impact on the end-user experience is minimal as going through an additional verification step every 3 months is acceptable for the user. Note that, in the absence of a mobile device, access to email can be used as a fallback mechanism.

• **Continuous authentication.** If so desired, our system can be used beyond the simple login process to protect against session hijacking without adding another usability burden for the client. We can use our mechanism to continuously authenticate the client because canvas renderings do not present changes in the span of a session and the authentication process is completely transparent to the user. Every few requests, we can recheck with our system the identity of the client's device and it has the benefit of hardening the work of an attacker wanting to defeat our scheme.

• **Managing separate devices.** It should be noted that each legitimate device has to be registered and authorized separately. This behavior is exactly the same as many websites which currently enforce 2FA. A simple analysis of the user-agent with other finer-grained attributes is sufficient to distinguish devices belonging to the same user. During the registration process of a new device, our canvas mechanism is bootstrapped so that it can be used in subsequent connections for verification. For each authorized device, our system will store a separate series of canvas tests to account for each device's different software/hardware layers.

3.3 Generation of unique canvas challenges

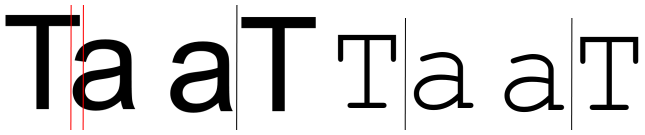
Section 2.1 showed that large canvas renderings with strings and color gradients are the prime candidates to exhibit diversity between devices. Shadows and curves are also small enhancements that can be utilized to further increase the gap between devices. Building on these results, we detail how we use the canvas API to generate unique challenges (i.e., unique test parameters) at each connection.

Fig. 7: *Example of a canvas test*

Figure 7 shows an example of a generated canvas images. It is composed of random strings and curves with randomly-generated gradients and shadows. The set of parameters that we randomize are defined below.

• **String Content.** Each font has its own repertoire of supported characters called glyphs. We use basic alpha-numerical glyphs that are guaranteed to be supported by most fonts. The generated strings in our challenges are 10 characters long with any combination of glyphs. Moreover, the order of letters in a string can be important and not all fonts behave the same way. Glyphs in monospaced fonts share the exact same amount of horizontal space while glyphs in proportional fonts can have different widths. Each proportional font contains a kerning table that defines the space value

between specific pairs of characters. OpenType fonts also support contextual kerning which defines the space between more than two consecutive glyphs.



(a) Arial *(b) Courier New*

Fig. 8: *Spacing comparison between fonts*

Figure 8 illustrates the kerning mechanism. With Arial which is a proportional font, the kerning table specifies a negative space value for the pair "Ta" but nothing for the pair "aT". For Courier New which is a monospaced or fixed-width font, each string occupies the exact same amount of horizontal space. In the end, this difference of spacing helps us increase the complexity of our tests.

• **Size and rotation of strings.** Our experiments detailed in Section 2 show that larger canvas renderings are better at distinguishing devices than smaller ones, because they are more precise and finer-grained. Also, as shown in Figure 9, rotating strings leads to pixelation and requires partial transparency to obtain smooth edges. In our case, a larger image leads to softer but well-defined limits between pixels.

(a) Zoom on a 'T' character before rotation *(b) Zoom on a 'T' character after rotation*

Fig. 9: *Details of a letter 'T' showing pixelation after rotation*

• **Curves.** While the impact of curves or rounded forms is definitely smaller than the one with strings, they can still provide an added value to our mechanism. Indeed, the shape of letters in a font can be straight with sharp and squared corners. Relying only on these letters can prevent us from displaying rounded strokes that generate graphical artifacts from aliasing effects. To increase the complexity of our canvas renderings, we generate cubic and quadratic Bezier curves. A curve is defined by starting and ending points along with a variable number of intermediate ones.

• **Color gradient.** With the introduction of CSS3, browsers now support linear and radial color gradient. A gradient can be seen as a natural progression from one color to the other and a CSS gradient element can be used on any string to change its color. An unlimited number of points can be defined in a gradient and every single one of these points can have its own color from the whole color palette. However, the main challenge is to find the right balance between the size of the string and the number of colors. Indeed, if the size of the rendered string is too small and the gradient is comprised of thousands of different colors, the rendering will not be large enough for all transitions to be visible, resulting in a meaningless cluster of pixels.

• **Number of strings and curves.** All the strings and curves in a canvas test have their own styles from their color gradient to their shadow. The overall complexity

of our test can be increased by generating dozens of strings and curves without impacting the performance of the authentication mechanism.

• **Shadow.** A shadow is defined by its color and the strength of its blur effect. A small strength value will cast a thin shadow around the rendered object but a higher value will disseminate a small cloud of pixels all around it (see Figure 10).

Fig. 10: *Identical forms with different shadow blurs (strongest blur on the right)*

3.4 Implementation

We developed a prototype of our mechanism in JavaScript, measuring both the performance and the diversity of generated challenges.

• **Steps and performance.** The first step of our prototype generates a random challenge for the client which includes the exact content, size, position and rotation of all the strings along with specific values for shadows and curves. The average generation time on a modern computer is less than 2ms. As a second step, the client generates the corresponding canvas rendering from the received challenge. Depending on the browser used, this step can take as much as 200ms for the most complex challenges. The average rendering time for Chrome users is about 50ms while it is around 100ms for Firefox users. Finally, since we collect a textual representation of an image, complex challenges can result in really long strings with more than 200,000 characters. To prevent storing these long strings in the authentication database, it is possible to use a cryptographic hash function like SHA-256 to hash all received responses. The stored values are then reduced to 256 bits while providing a 128-bit security against collisions which is more than sufficient for the data we are handling. The recorded average hashing time is around 40ms. Overall, the complete process takes less than 250ms which is clearly acceptable for an authentication scheme.

• **Evaluation.** To ensure that our mechanism is supported by most browsers, we deployed our script alongside the one used for our test described in Section 4.5. We generated a unique challenge for more than 1,111,000 devices and collected the associated response for each of them. We also checked that two challenges with supposedly different parameters would not produce two identical renderings (e.g., in the case that two challenges are different on paper but translate into identical forms and figures in the final image). The analysis shows that 99.9% of devices returned a canvas rendering and all collected images were unique. The 0.1% of non-unique values come from either older browsers that do not support the canvas API and returned identical strings, or browsers with extensions that blocked the rendering process and returned empty canvases. This result shows that the canvas API is supported by most devices and it gives us confidence that the challenges generated by our mechanism exhibit enough diversity.

4 Security analysis

In this section, we define the adversary model and analyze the security provided by our authentication system against known attacks.

4.1 Adversary model

The goal of the adversary is to impersonate a client (a user's browser) and fool the authentication server into believing that the sent images come from the client. With this goal in mind, we define the adversary model as follows:

- The adversary cannot tamper with the server and modify its behavior.
- The adversary cannot eavesdrop messages exchanged between the server and the client. We ensure this by setting up a confidential tunnel for every connection.
- The adversary cannot interfere with the very first exchange between the server and the client. As with any password-based online authentication system, the first exchange must be authenticated by another channel, e.g., a confirmation email during the registration step.
- The adversary does not know the client's device configuration, and neither does the server. By analogy with keyed cryptographic functions, the device configuration is the client's secret key.
- The adversary knows the code to generate a canvas rendering from a challenge.
- The adversary can set up a fake server and query the client with a polynomially-bounded number of chosen challenges.

4.2 Replay attack

• **Example.** An attacker listens to the Internet traffic of the victim and eavesdrops on a genuine connection between the server and the client. His goal is to collect the victim's responses to the canvas challenges to replay them in a future connection.
• **Analysis.** Before the protocol defined in Section 3.1 is executed, we assume that a secure channel is established between the server and the client. This can be easily achieved, for example with TLS, which is widely deployed on the web. Given the properties of a secure channel, a replay attack cannot occur.

4.3 Website MITM or relay attacks

• **Example.** An attacker creates a phishing website that masquerades as a legitimate one. To get access to the service, victim users will enter their credentials not knowing that the data is directly transmitted to the attacker. The fake server can then ask the victim's browser to compute any canvas challenges while the connection is active and is therefore capable of replaying the benign server's canvas challenges.
• **Analysis.** Introduced in 1976 by Conway with the Chess Grandmaster problem [14], relay attacks cannot be avoided through classical cryptographic means. As such, our canvas authentication mechanism, in and of itself, cannot stop attackers who, through their phishing sites, replay in real time, the canvas challenges presented by the legitimate server post password authentication.

At the same time, we want to point out that even with this limitation, our proposed canvas authentication solution significantly complicates attacks and makes them more costly to conduct and more fragile. First, a MITM attack requires the victim to be online when performing the login process which greatly reduces the window of vulnerability. A phishing website can no longer merely collect the credentials from users and deliver them to the attacker for later usage or reselling. The phishing website *must* connect, in real time, to the benign server being mimicked, log in, and start relaying canvas challenges. Even though this is technically possible, an attacker cannot conduct these attacks in large scale without a significant investment in resources. Too many connections from the phishing website for too many users will trigger other intrusion-detection systems at the benign site, as will mismatches between the geolocation of the benign user being phished and the location of the relaying phishing website. To bypass these limitations, attackers must set up a complex network of proxies to provide load-balancing and appropriate geolocation all of which have to operate in real time when a user is visiting a phishing page. Even if attackers successfully collect user credentials, unless they are able to relay canvas challenges and bypass other intrusion detection systems *before* a user closes their tab, they will have failed in authenticating to the protected service. Second, because the victim needs to be online and on the attacker's phishing page, attackers can no longer conduct successful credential brute-forcing attacks or abuse reused credentials from password leaks on other services. Even if the attackers discover the appropriate credential combination, in the absence of the user's browser, they will be unable to construct the canvas fingerprint expected by our system.

Overall, we argue that, even though our proposed system is theoretically vulnerable to MITM attacks, in practice it limits and significantly complicates attack scenarios, which immediately translates to reduction in successful account hijacking attempts.

4.4 Preplay attack

• **Example.** An attacker sets up a webpage to collect ahead of time any canvas renderings he desires from the victim's device. The particularity of this attack is that it can be completely stealthy as it does not require user's interaction. Moreover, the attacker does not necessarily need to put some effort into building the perfect phishing opportunity as any controlled webpage can run the desired scripts.

• **Analysis.** In a preplay attack, an attacker queries the client with arbitrary challenges, expecting that the challenge that will be sent by the server will belong to the arbitrary challenges selected by the attacker. This way, an attacker has the necessary tools to correctly get through our authentication mechanism. In this section, we consequently analyze how likely and practical it is for the attacker to obtain responses and whether they allow him to perform an attack with a non-negligible probability.

Injectivity of the canvas rendering process. For a given device, different challenges produce different canvas renderings because the rendering process is injective. Indeed, every single parameter that is described in Section 3.3 has an impact on a canvas rendering. Positioning a string and choosing its size will define which pixels of the canvas are not blank. Generating a gradient will modify the RGB channels of each pixel of a string. Rotating a string will use partial transparency to have a faithful

result and to define precisely the limits between pixels. In the end, no two challenges will produce the same response on a single device. Even the smallest change of color that the eye cannot perceive will have an impact on the final picture.

Exhaustive collection of responses. Since the actual code of our canvas rendering process is not secret, the attacker can set up a fake authentication server that poses as a legitimate one and asks the victim any challenge he wants. In other terms, this means that an attacker has a partial access to the victim's device to collect any responses he desires. Here, we estimate the number of possible challenges and the time it would take to transfer and store all responses. It should be noted that we do not consider the use of curves for this calculation. Estimating the number of challenges:

- String content: Generated strings are composed of 10 alpha-numerical characters. We consider both lower-case and upper-case variants of standard letters. In total, we have 26 upper-case letters, 26 lower-case letters, and 10 figures. We so have 62^{10} combinations.
- Size: Bigger font sizes lead to a better distinction of devices. We fix the lower bound at 30 and the upper one at 78, which leads to 49 different font sizes.
- Rotation: Every string can be rotated by any specific value. Following tests we performed, a precision of the rotation up to the tenth digit has an impact on the canvas rendering. Smaller variations do not result in detectable changes. We consequently consider $360^o \times 10 = 3600$ different rotations.
- Gradient: The two parameters of a gradient are its colors and the position of each of these colors. The RGB color model is used and each color is encoded on 8 bits so we have 2^{24} different colors at our disposal. We use the "Math.random()" JavaScript function to give us the position of each color on the gradient line. This function returns a number between 0 and 1 and it has a 52-bit precision. Variations from the thousandth digits have seemingly little to no impact because of the limited number of pixels in our images. We only consider precision up to the hundredth digit and we limit the number of different colors in a given gradient to 100 with a lower bound at two (one color at each extremity of the gradient line). Considering two colors provide a conservative lower bound on the number of different gradients, we have $(2^{24})^2 = 2^{48}$ combinations.
- Shadow: The color and the blur of the shadow can be tweaked in the canvas API. The selection of the color is identical to the one described for the gradient so it provides 2^{24} possibilities and we constrain the strength of the blur between 0 and 50.
- $Total = 62^{10} \times 49 \times 3600 \times 2^{48} \times 2^{24} \times 51 \approx 2^{154}$ challenges

Taking into account practical implications, storing all responses would occupy 2.3×10^{50} bits with an average of 10 kb per response. It would take several quintilliard years on a Gigabit internet connection to transfer everything without considering possible network instabilities and congestion. The sheer size of these numbers eliminates all possibilities to conduct a successful attack following this approach.

4.5 Guessing or building the right response

• **Example.** An attack against a challenge-response protocol consists for an attacker to guess or build a valid response upon reception of a canvas challenge. Here, the

attacker may have access to a cluster of compromised devices to ask for canvas renderings and he may also have specific techniques to build a response from previously observed renderings.

• **Analysis.** To defeat such an attack, the set of possible responses should be large enough, and the rendering process should be non-malleable.

Blind guess. An attacker could blindly generate an image regardless of the sent challenge. This way, he can set any desired RGBA values for all the pixels of the response. Since the canvas size in our mechanism is 1900x300, the total number of pixels is 570,000. Given the alpha value is not random (it strongly depends on the image content), a conservative approach consists in not considering it in the analysis: the number of possible responses is then $2^{24\times 570000} = 2^{13680000}$ which is far too high to consider this approach feasible.

Choosing the most popular response. With the help of our partner who handles one of the top 15 French websites (according to the Alexa traffic rank), we sent the exact same challenge to more than 1,111,000 devices on a period of 42 days between December 2016 and January 2017. Devices who visited the weather or politics page of the official website of this partner received this test. To be compliant with current data collection laws, we only collected the response of visitors who consented to the use of cookies. It should be noted that this test is different from the one described in Section 3.4 where we generated a unique challenge for each device. Here, we sent the same challenge to everyone. In total, we observed 9,698 different responses, and 4,645 responses were received only once (i.e., a single device answered this response). Figure 11 shows the distribution of sets that share the same responses. 98.7% of them contain each less than 0.1% of the population. Only a single set is above the 5% threshold and it represents 9.9% of the population.

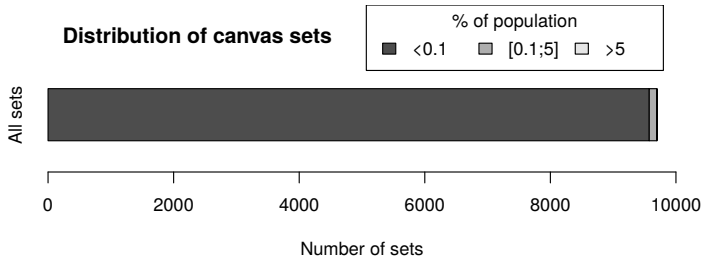


Fig. 11: *Number of sets containing x equal responses*

The attacker aims to send the same response than the one from the victim he seeks to impersonate. To do so, the attacker can request his own computer with the challenge received from the server. The attack succeeds if and only if the response generated by the attacker matches the victim's expected response. Calculating the success probability is related to the birthday paradox on a non-uniform distribution. Assuming p_i is the number of responses received in the set i ($1 \leq i \leq 9698$) divided

by the total number of received responses, the attacker's success probability is:

$$p = \sum_{i=1}^{i=9698} p_i^2.$$

Given the numerical values collected in our experiments, we have $p < 0.01$. Note that the attacker can increase his success probability if he knows the distribution provided in Figure 11. Indeed, to maximize his probability of guessing the right response, the attacker would choose the response sent by the highest number of devices. The bigger this set is, the higher his probability of fooling the system will be. Obtaining the full distribution of responses is in fact not required to maximise the probability of success: identifying the largest set is enough. To do so, the attacker can use a cluster of computers with common configurations (e.g., a botnet): upon reception of a challenge from an authentication server, the attacker sends this challenge to each computer of the cluster and can expect to identify the most common canvas renderings for a given challenge. It is worth noting that this attack requires a cluster and allows the attacker to increase his success probability up to 9.9% for this particular challenge. The obtained distribution will vary depending on the challenge and its complexity. There is also no guarantee that a device present in the biggest cluster will stay in it with a different challenge, which makes this approach difficult to use.

Forging canvas renderings. Instead of guessing the response, an attacker can try to recreate a genuine response from observed ones. Although the rendering process is malleable to some extent, our experiments show that its malleability is quite limited. In particular, we consider the following actions possible: resizing glyphs, changing the place of glyphs in a string, rotating glyphs, and applying a custom gradient. We saw in Section 3.3 that the order of glyphs is important in proportional fonts because of kerning. If an attacker were able to learn all combinations of two letters ($52^2 = 2704$), he would not need to enumerate all the strings of 10 characters so the difficulty of knowing all responses would be lowered. On the other hand, changing font, size, and rotating strings increase the difficulty of the attack. These operations generate distortions and create artifacts like aliasing or blurring. Interpolation between pixels must be perfect so that a rendering is faithful to an original one. Moreover, with the use of gradients and shadows, the complexity of our test increases since it becomes even harder to find the right balance between colors and transparency to achieve a perfect forgery.

Defining precisely what is possible through image modification is still an open question. We consider though that the number of involved parameters makes the attack hardly achievable even with the help of strong image transformation tools. Although simple cases can occur, e.g., a non-rotated string with the same letter 10 times and a gradient with the same color on both ends, the probability that such a weak challenge is randomly picked is negligible. In the end, forging canvas renderings does not seem to be a relevant attack because guessing the response is an easier attack, with a higher success probability.

4.6 Protection against configuration recovery

• **Example.** By getting the browser's fingerprint of the victim from a controlled webpage, the attacker can try to rebuild the same configuration or even buy the same device to start from a configuration that is as close as possible to the real one.
• **Analysis.** The last attack is the configuration recovery, which is somehow equivalent to a key-recovery attack in a classical cryptographic scheme. The full knowledge of the configuration of a client is indeed enough and sufficient to answer correctly to any challenge. Contrarily to classical cryptography, though, the key is not a 128-bit binary secret but the full hardware and software configuration of the client. Partial knowledge about the configuration can be obtained by the adversary using a browser fingerprinting script. This mechanism indeed provides the attacker with information on the targeted victim, e.g., the browser model, the operating system, and the GPU model. A key issue consists in evaluating how much the device configuration leaks when the client faces a browser fingerprinting attack. We need to evaluate whether a browser fingerprint provides the attacker with enough information to build valid responses. Note that current fingerprinting techniques do not reveal the full device's configuration, e.g., they can not catch the device's driver, kernel, and BIOS versions, to name but a few.

Our analysis considers the set of fingerprints used in the previous section, and we divided our dataset into two categories: desktops and mobile devices. This distinction is important because desktops are highly customizable whereas smartphones are highly standardized and present a lot of similarity across models and brands. 93.3% of these 1,111,819 fingerprints come from desktop devices and 6.5% from mobile ones. Less than 0.2% are discarded because they either come from bots or are devices that could not be identified (their user-agents did not give enough information). In order to determine the advantage a user can get, we regrouped fingerprints that were identical with each other and we looked at the generated responses. The collected fingerprints were composed of the following attributes: the HTTP user agent header, the HTTP language header, the platform, the CPU class, the WebGL renderer (GPU of the device), and the width, height and color depth of the screen. Inside the same group, if the canvas renderings are different from each other, this means that the fingerprints do not capture the totality of the client's configuration (i.e. the key is partial). An attacker would then not be able to recreate faithfully the victim's renderings even if he had the same fingerprint.

Figure 12 shows the distribution for desktop and mobile devices. For each category, the distribution is divided into three: groups with a single rendering (i.e. all the devices in the group have generated the exact same image), groups with 1 to 5 renderings, groups with more than 5 different renderings.

Desktop. With the heterogeneity of desktop configurations, we perform the analysis by keeping all the fingerprint attributes. About 57.9% of groups have devices that share the same canvas rendering. This number is pretty high but it does not give an attacker full confidence that he will produce the right rendering even if he has the same fingerprint as his victim.

Mobile. On mobile devices, one can identify the type and the model of the device. As detailed in [20], "some smartphones running Android give the exact model and firmware version of their phone" via the user-agent. Since phones with the same

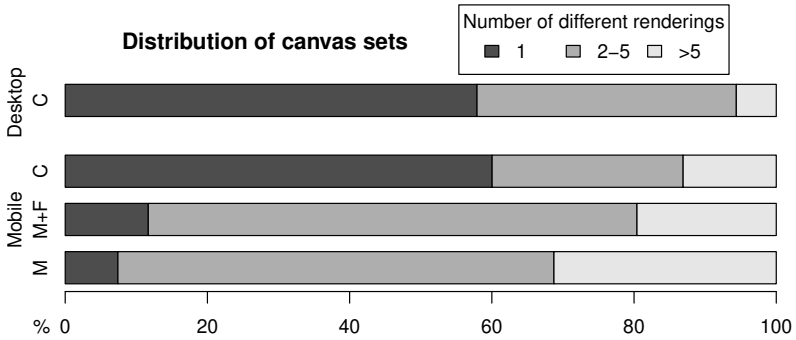


Fig. 12: *Distribution of canvas renderings for groups with identical fingerprints (C=Complete, M=Model, F=Firmware)*

model have the same specifications, we must verify whether all of them present identical renderings, e.g., evaluate whether the version of the Android firmware has an impact on the canvas painting process. To find the answer, we decided to adjust the information contained in a fingerprint and created three different categories:

- Complete: We took the complete fingerprints like we did for the desktop analysis.
- Model and firmware: We extracted from the user agent the model and the firmware of the device.
- Model: We limited ourselves to the phone model only.

First, if we consider all the collected attributes, the percentage of groups with a unique rendering is 60%. This value is higher than what is observed with desktops, which was expected since mobile devices are a lot less customizable. Then, if we limit ourselves to the model and its firmware, the groups with a unique rendering drops to 11.7%. This significant drop can be explained by the fact that software customization has an impact on the canvas rendering. Notably, there are many different apps and browsers available that can influence the generation of the response. Finally, if we only identify the model of the phone, it proves to be insufficient for an attacker since the percentage drops to a meager 7.4%. This means that buying the same smartphone as the victim still requires some work to be able to faithfully replicate a canvas rendering. It is really surprising to see that there can be a lot of diversity even when considering the exact same phone with the exact same firmware. These numbers demonstrate that our mechanism is resilient to configuration recovery on both desktops and smartphones.

In the end, even if the knowledge of a browser fingerprint makes easier a configuration-recovery attack, the problem can be mitigated if the client does not fully reveal his fingerprint. It is also worth noting that this attack requires extra work as the attacker has to set up a computer whose fingerprint is the same as the one of his victim, or manage a cluster of computers whose at least one possesses the expected fingerprint.

5 Canvas Fingerprinting vs. User Privacy

Due to the association of browser fingerprinting with privacy-intrusive practices on the web, one may wonder whether our proposed canvas-based authentication scheme

compromises, in some way, a user's privacy. Similar to prior work [25], we argue that browser fingerprinting is intrusive not because of websites obtaining attributes of a user's browser, but because third parties which obtain these attributes from multiple websites, can *link* user visits together and therefore track the user across unrelated websites and infer sensitive information about them. Contrastingly, the canvas-based authentication method that we propose in this paper is utilized by first-party websites as an extra layer of authentication. These websites already utilize first-party cookies in order to track users and their authentication status. As such, a first-party canvas fingerprint of the same user does not provide any additional linking power to these first-party websites. Since our protocol hinges on the secrecy of canvas challenges, if first-party websites shared the collected canvas fingerprints with others, they would be directly weakening the security of their own authentication system. Instead, different websites utilizing our proposed canvas-based authentication scheme will be storing their own different canvas fingerprints for each user with no incentive for linking them together. Overall, we are confident that our proposed canvas-based authentication system increases a user's account security, without compromising any of their privacy.

6 Related work

Browser fingerprinting. Introduced by Eckersley in 2010 [16], browser fingerprinting has attracted several studies which have shown both the ease to collect a fingerprint [26, 20, 18] and the deployment of fingerprinting scripts on the Internet [9, 8, 17]. Different techniques have been discovered to reinforce identification by querying unexplored parts of the system [24, 23]. Recent studies have looked more specifically at the fingerprintability of browser extensions [35, 30, 27, 34]. Finally, simple heuristics [16] can be combined with more advanced methods [37] to track a fingerprint's evolution.

Canvas fingerprinting. Discovered by Mowery et al. [23] and investigated by Acar et al. [8], canvas fingerprinting consists in using the HTML canvas element to draw strings and shapes in the browser. A study by Laperdrix et al. shows that canvas fingerprinting is in the top 5 of the most discriminating attributes and that it plays an essential role in identifying browsers on smartphones [20]. Researchers at Google explored the use of canvas fingerprinting to identify *device classes*, e.g. operating systems and browsers, with a system called Picasso [12]. Their goal is to filter inorganic traffic (spoofed clients, emulated devices) to fight internet abuse by bots and scripts. Finally, Vastel et al. designed a scanner to detect fingerprint inconsistencies [36].

Authentication with browser fingerprinting. Spooren et al. [33] focus on the fingerprinting of mobile devices for risk-based authentication. Their main observation is that it is easy to for an adversary to set up a web page in order to collect browser fingerprints and then use them to impersonate the users. Our work shows that fingerprinting with dynamic queries, such as canvas fingerprinting, addresses the risks observed in the work of Spooren et al., and can be used as an effective additional factor for authentication. Alaca et al. provide an extensive yet purely theoretical analysis of current fingerprinting techniques for augmenting web authentication [10]. To the best of our knowledge, we are the very first to provide an in-depth quantitative analysis of the protection provided by a dynamic fingerprinting scheme.

7 Conclusion

In this paper, we presented the first constructive use of browser fingerprinting that is not susceptible to simple replay attacks. We designed a challenge/response protocol based on canvas fingerprinting and showed how this mechanism can differentiate between legitimate users and those impersonating them. We evaluated our protocol against a wide range of attacks showing that it completely invalidates certain classes of attacks, such as, credential brute-forcing, while it significantly complicates the exploitation for others. We quantified the stability of canvas fingerprints for legitimate users and identified the canvas attributes that generate the most diverse, software/hardware-dependent canvas fingerprints, showing that our canvas-based authentication protocol can be straightforwardly utilized by websites.

Acknowledgements: We thank our shepherd Deborah Shands and the anonymous reviewers for their helpful feedback. This work was supported by the Office of Naval Research (ONR) under grant N00014-16-1-2264 and by the National Science Foundation (NSF) under grants CNS-1813974, CMMI-1842020, CNS-1617593, and CNS-1527086.
Availability: Additional results on our design phase along with code and demo pages can be found at: `https://github.com/plaperdr/morellian-canvas`.

References

1. Pale Moon browser - Version 25.6.0 adds a canvas poisoning feature (2015), `https://www.palemoon.org/releasenotes-archived.shtml`
2. Bugzilla - Bug 1231701: Ship an emoji font on Windows XP-7 (2017), `https://bugzilla.mozilla.org/show_bug.cgi?id=1231701`
3. Yahoo breach actually hit all 3 billion user accounts - CNET (2017), `https://www.cnet.com/news/yahoo-announces-all-3-billion-accounts-hit-in-2013-breach/`
4. Canvas Defender - Firefox add-on that adds noise to a canvas element (2018), `https://addons.mozilla.org/en-US/firefox/addon/no-canvas-fingerprinting/`
5. Over a billion people's data was compromised in 2018 - NordVPN (2018), `https://nordvpn.com/blog/biggest-data-breaches-2018/`
6. Two Factor Auth List - List of websites supporting two-factor authentication and the methods they use (2018), `https://twofactorauth.org/`
7. User Authentication with OAuth 2.0 (2018), `https://oauth.net/articles/authentication/`
8. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Diaz, C.: The web never forgets: Persistent tracking mechanisms in the wild. In: CCS'14
9. Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., Preneel, B.: FPDetective: dusting the web for fingerprinters. In: CCS'13
10. Alaca, F., van Oorschot, P.: Device fingerprinting for augmenting web authentication: Classification and analysis of methods. In: ACSAC'16
11. Bonneau, J., Herley, C., Van Oorschot, P.C., Stajano, F.: The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In: S&P'12
12. Bursztein, E., Malyshev, A., Pietraszek, T., Thomas, K.: Picasso: Lightweight device class fingerprinting for web clients. In: SPSM'16
13. Cao, Y., Li, S., Wijmans, E.: (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In: NDSS'17

14. Conway, J.H.: On Numbers and Games. No. 6 in London Mathematical Society Monographs, Academic Press, London-New-San Francisco (1976)
15. Duo Labs: State of the Auth: Experiences and Perceptions of Multi-Factor Authentication. https://duo.com/assets/ebooks/state-of-the-auth.pdf
16. Eckersley, P.: How unique is your web browser? In: PETS'10
17. Englehardt, S., Narayanan, A.: Online tracking: A 1-million-site measurement and analysis. In: CCS'16
18. Gómez-Boix, A., Laperdrix, P., Baudry, B.: Hiding in the crowd: An analysis of the effectiveness of browser fingerprinting at large scale. WWW'18
19. Laperdrix, P., Baudry, B., Mishra, V.: FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In: ESSoS'17
20. Laperdrix, P., Rudametkin, W., Baudry, B.: Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In: S&P'16
21. Laperdrix, P., Rudametkin, W., Baudry, B.: Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification. In: SEAMS'15
22. Milka, G.: Anatomy of Account Takeover (2018), https://www.usenix.org/node/208154
23. Mowery, K., Shacham, H.: Pixel perfect: Fingerprinting canvas in HTML5. In: W2SP'12
24. Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., Weippl, E., Wien, F.C.: Fast and reliable browser identification with javascript engine fingerprinting. In: W2SP'13
25. Nikiforakis, N., Joosen, W., Livshits, B.: Privaricator: Deceiving fingerprinters with little white lies. In: WWW'15
26. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In: S&P'13
27. Picazo-Sanchez, P., Sjösten, A., Van Acker, S., Sabelfeld, A.: LATEX GLOVES: Protecting Browser Extensions from Probing and Revelation Attacks. In: NDSS'19
28. Rupertus Fine Art Research: What Is Morellian Analysis. http://rupertusresearch.com/2016/09/27/what-is-morellian-analysis/ (2016)
29. Sánchez-Rola, I., Santos, I., Balzarotti, D.: Clock Around the Clock: Time-Based Device Fingerprinting. In: CCS'18
30. Sanchez-Rola, I., Santos, I., Balzarotti, D.: Extension breakdown: Security analysis of browsers extension resources control policies. In: USENIX Security'17
31. Shannon, C.E.: A mathematical theory of communication. Bell system technical journal **27**(3), 379–423 (1948)
32. Sjösten, A., Van Acker, S., Sabelfeld, A.: Discovering Browser Extensions via Web Accessible Resources. In: CODASPY'17
33. Spooren, J., Preuveneers, D., Joosen, W.: Mobile device fingerprinting considered harmful for risk-based authentication. In: EuroSec'15
34. Starov, O., Laperdrix, P., Kapravelos, A., Nikiforakis, N.: Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In: WWW'19
35. Starov, O., Nikiforakis, N.: XHOUND: Quantifying the Fingerprintability of Browser Extensions. In: S&P'17
36. Vastel, A., Laperdrix, P., Rudametkin, W., Rouvoy, R.: FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In: USENIX Security'18
37. Vastel, A., Laperdrix, P., Rudametkin, W., Rouvoy, R.: FP-STALKER: Tracking Browser Fingerprint Evolutions. In: S&P'18