Phish in Sheep's Clothing: Exploring the Authentication Pitfalls of Browser Fingerprinting

Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis
University of Illinois at Chicago, {xlin48, pilia, ssolan5, polakis}@uic.edu

Abstract

As users navigate the web they face a multitude of threats; among them, attacks that result in account compromise can be particularly devastating. In a world fraught with data breaches and sophisticated phishing attacks, web services strive to fortify user accounts by adopting new mechanisms that identify and prevent suspicious login attempts. More recently, browser fingerprinting techniques have been incorporated into the authentication workflow of major services as part of their decision-making process for triggering additional security mechanisms (e.g., two-factor authentication).

In this paper we present the first comprehensive and in-depth exploration of the security implications of *real-world* systems relying on browser fingerprints for authentication. Guided by our investigation, we develop a tool for automatically constructing fingerprinting vectors that replicate the process of target websites, enabling the extraction of fingerprints from users' devices that *exactly* match those generated by target websites. Subsequently, we demonstrate how *phishing* attackers can replicate users' fingerprints on *different* devices to deceive the risk-based authentication systems of *high-value* web services (e.g., cryptocurrency trading) to *completely bypass two-factor authentication*. To gain a better understanding of whether attackers can carry out such attacks, we study the evolution of browser fingerprinting practices in phishing websites over time. While attackers do not generally collect all the necessary fingerprinting attributes, unfortunately that is not the case for attackers targeting certain financial institutions where we observe an increasing number of phishing sites capable of pulling off our attacks. To address the significant threat posed by our attack, we have disclosed our findings to the vulnerable vendors.

1 Introduction

The web plays a pivotal role in many facets of everyday life, and ensuring that user accounts and the sensitive data found therein are protected is of paramount importance. Consequently, account hijacking attacks pose a serious threat to users. In fact, OWASP reports *broken authentication* as the second biggest risk in its list of top web application risks [11]. Additionally, phishing remains the most common account compromise vector for major web services [23, 68].

To prevent unauthorized access, online services often employ various forms of risk-based authentication mechanisms, which attempt to identify suspicious login attempts [74] (e.g., based on the geolocation of the IP address [58]). When such attempts are detected, services can take different courses of action, with two-factor authentication (2FA) being the de facto defense for fortifying accounts against hijacking attacks. In most cases 2FA significantly raises the bar, as attackers will also need to obtain the second factor. Indeed, a study of over 350K real-world hijacking attempts against Google accounts found that device-based 2FA blocks more than 94% of the attempts that originate from phishing attacks [30].

However, due to the usability challenges and friction [73] introduced by 2FA [60], sites may attempt to minimize the inconvenience by reducing the frequency of 2FA challenges so that they are only shown when a login attempt presents some risk. If the site determines that the user entering the credentials is indeed the account owner, it will not trigger 2FA. To achieve that, sites gather information about the user's "environment" during the authentication process. More recently, as browser fingerprinting [46] has gained significant traction as a web tracking vector [16], such device fingerprints have been incorporated into the risk analysis process so as to augment authentication. During the login workflow websites can collect the user's device fingerprints (e.g., installed fonts, canvas rendering) and compare them to the fingerprints collected during a previous legitimate browsing session. If the values match, this is a strong indication that the login attempt is legitimate and the website can refrain from showing a 2FA challenge.

Unfortunately, the inherent characteristics of browser fingerprints that render them an attractive authentication-augmenting factor also lend them to being used against the authentication process itself. This dichotomy forms the key motivation behind our research. Specifically, browser fingerprints can be trivially generated by *any* website the user visits through a series

of JavaScript functions, and certain fingerprinting attributes remain stable over time [71]. Accordingly, malicious websites can collect users' fingerprints and use them to mimic the user's device, thus negating the protection offered by additional authentication mechanisms. While many attacks can lead to account hijacking (e.g., credential stuffing [69], cookie hijacking [36, 62, 63]) our main threat model is that of an attacker that has deployed a phishing website, which the victim user visits.

In this paper we provide the first, to the best of our knowledge, empirical *real-world* analysis of the security implications of leveraging browser fingerprints for augmenting the authentication process. Specifically, we provide an in-depth investigation of how high-value web services rely on browser fingerprints, and detail how attackers can generate and misuse a user's fingerprints to completely bypass two-factor authentication. First, we explore popular websites across different high-value categories (e.g., banking, tax-related) and identify those that leverage fingerprints during the login process, using a combination of static and dynamic analysis. Guided by our findings, we build a browser extension that identifies and "extracts" the fingerprinting process deployed by a website and automatically generates the code that will calculate the exact fingerprint that the website would create for a specific device. This includes techniques like canvas fingerprinting, which require specific input data and parameters to produce an identical fingerprint. We then incorporate the fingerprinting code of targeted websites into a testing website; when a user visits the website it generates the user's fingerprint for each target website. Subsequently, we visit each website from a *different device* (i.e., the attacker's device) and try to log into the user's account by providing the correct credentials. To mimic the user's device, we develop an extension that loads the user's corresponding fingerprints for a given website and mimics all the fingerprinting attributes that the target website collects. This manipulates the website into considering the attacker's device to be a *known* device and deciding that 2FA should not be triggered.

We experimentally evaluate our attack against popular and highly-valuable websites across different categories and provide an in-depth analysis and assessment of the inner workings of risk-based authentication systems. While our study reveals that reliance on browser fingerprints as a strong signal during the authentication process is not yet widespread, it is common among *critical* and *high value* services (e.g., financial). We are able to bypass the security offered by risk-based authentication in ten of the 16 (62.5%) popular services we tested (including a bank, a credit card company, and a cryptocurrency trading service). In practice, our attack is only ineffective against a subset of the services that require the attacker to obtain an IP address used by the victim in the past. Finally, we conduct an extensive exploration of the phishing ecosystem and identify rampant deployment of browser fingerprinting, including advanced techniques. Our analysis reveals that phishers already collect sufficient information to carry out our attacks in a subset of the analyzed websites.

Overall, as account hijacking remains a serious threat to users, 2FA is a crucial component of account protection and is typically the *last line of defense*. While using browser fingerprints for augmenting authentication can prevent certain attacks like credential stuffing, our research demonstrates that it can also have the *opposite* effect and *undermine* an account's security by allowing attackers to *completely bypass* any 2FA mechanism. Naturally, this poses a significant threat to users. As such, we present the first comprehensive and in-depth assessment of the security implications of ancillary authentication mechanisms, and find that fingerprints *must* be used *in conjunction* with other signals (e.g., IP address, requiring the presence of specific cookies) to prevent our attack. We hope that our findings will kickstart additional research on the security implications of browser fingerprinting and further incentivize studies on more robust fingerprinting techniques for web authentication. In summary, our research contributions are:

- We present a novel practical attack that demonstrates how phishers can obtain and replicate users' *exact* browser fingerprints to deceive risk-based authentication mechanisms and eliminate two-factor authentication. We develop an automated pipeline for extracting and constructing website-specific fingerprinting vectors, as well as mimicking a target user's browser in the targeted websites.
- We present an empirical exploration and in-depth analysis of the use of browser fingerprinting for augmenting web authentication in the wild. Our experimental evaluation reveals that while this is not yet a widespread practice in general, it is prevalent in high-value services, highlighting the severe implications of our attack.
- We present a large-scale study on the use of browser fingerprinting techniques by phishing sites. Our findings reveal that this has become an increasingly common phenomenon, with the majority of phishing sites gathering browser fingerprints. Alarmingly, we identify a series of sites generating all the fingerprinting attributes necessary for bypassing 2FA in major financial institutions.
- In an effort to kickstart remediation, we have disclosed our findings and offered remediation guidelines to affected vendors. Additionally, we will share our code with other researchers to foster more research in the area.

2 Background and Threat Model

Here we briefly present pertinent background information and then define the threat model that will guide our analysis.

Browser fingerprinting. Cookie-based tracking has been an unavoidable aspect of web browsing for more than two decades [33, 34, 48]. However, with average users becoming more privacy-aware and browsers deploying anti-tracking defenses, cookie-less tracking techniques have come to the forefront [53]. In fact, browser fingerprinting has garnered significant attention from the security community, resulting in a multitude of measurement studies and fingerprinting techniques

being demonstrated [25, 32, 38, 40, 43, 45, 46, 52, 64, 66, 71]. In a nutshell, browser fingerprinting is a stateless approach to tracking users, which identifies a series of attributes that possess "discriminating" power, thus allowing websites to re-identify users based on unique characteristics of their browser and device. All these attributes, including those for more advanced fingerprinting techniques, can be obtained through various JavaScript APIs available in modern browsers.

Threat model. Studies have reported that phishing remains the most common source of account hijacking, even in major services [23, 68]. Preventing phishers and other attackers that have obtained users' credentials (e.g., through keyloggers) from gaining access to users' accounts is a significant authentication challenge, which has contributed to the rise of risk-based authentication mechanisms. In our main attacker model we assume that the attacker is able to trick the user into visiting a malicious website and divulging their credentials. The malicious website includes JavaScript code that generates the exact browser/device fingerprints that the target website would generate for that specific user (we describe how we achieve this in §3.1). To increase the attack's coverage, the website also generates site-specific fingerprints for a series of additional target websites, so as to take advantage of any instances of the password being reused [26, 57].

While our main threat model is that of a phisher, due to the fact that risk-based checks typically occur at login time, a large-scale study on cookie-hijacking attacks [31] reported a website that prevented cookie hijacking by also employing these checks on requests carrying valid authentication cookies from pre-established sessions. As such, in our evaluation we also include an experiment that investigates this specific scenario. It is, however, important to note that since the attacker needs the ability to execute JavaScript code to obtain the user's browser fingerprints, this attack is limited to attackers that can execute arbitrary code *and* also steal the user's cookies (e.g., through an XSS attack [47]), thus precluding passive network eavesdroppers. Moreover, attacks against password managers and autofill [49, 54, 61] could also potentially be used, but we do not explore such attacks in this work.

We emphasize that our threat model focuses on attackers that are able to obtain the user's browser fingerprints by luring them to a malicious website, and then submit them from *their own* device. Our attack is not applicable to other attacks (e.g., brute-forcing, credential-stuffing that relies on passwords leaked in data breaches, etc.). Similarly, if the user visits the phishing website from a different browser or device than the one normally used to log into their accounts, the use of browser fingerprints as a risk-based authentication signal would prevent the phishing attacker from compromising the user's accounts.

3 Misusing Browser Fingerprints

We present a novel attack that allows an attacker to bypass 2FA by deceiving web services into thinking that the attacker's login attempt originates from a *known* device, i.e., the user's device. In essence, our attack leverages the use of browser fingerprinting techniques as a part of the risk-based authentication process. An overview of our attack is shown in Figure 1. In this section we give an overview of our methodology and then describe in detail how we "extract" the fingerprinting code from vulnerable websites (*phase 1*) which is incorporated into our attack site (*phase 2*). Finally, we detail the spoofing process when the attacker attempts to log into the victim's accounts (*phase 3*).

Attack overview. The main observation behind our attack is that sites that employ 2FA often "remember" users' devices during authentication, so as to minimize friction and skip 2FA for legitimate login attempts. In other words, if the user has visited these websites in the past from that specific device, they consider the login attempt to be low-risk and refrain from asking the user to solve a 2FA challenge. Through an initial manual exploration we observed that these websites often rely on HTTP cookies but may also generate and collect browser fingerprints (and in some cases rely on a combination of both) for determining whether they have encountered the device in the past. However, since these fingerprints can be easily collected using JavaScript, our work explores whether we can circumvent authentication mechanisms that rely on browser fingerprinting.

For our experiments we built a honeysite that plays the role of the attacker's phishing site and serves the purpose of collecting users' fingerprints. Our honeysite does not rely on a generic fingerprinting library, such as `Fingerprintjs2` but, instead, *tailors* the fingerprint-generation process to match that of a targeted website. For example, if the attacker deploys a phishing site that aims to steal the user's Google credentials, the site should also replicate Google's exact fingerprint-generation process. To that end, in the preparatory phase of our attack (Figure 1 – *phase 1*) we visit the target websites and "extract" their fingerprinting code. That specific code is then incorporated in our honeysite. The reasoning behind this is that any differences in the fingerprinting implementation can result in us generating values that are different from those that the target website expects from the user. For instance, our implementation can use different parameters when computing advanced fingerprints (e.g., render different images with WebGL or enumerate a different list of fonts), which would result in different fingerprints.

By *automatically* analyzing and replicating the fingerprinting process of target websites, we ensure that the fingerprints we collect are generated identically to the ones expected by the target websites. We note that an attacker can incorporate the fingerprinting code of multiple sites in the phishing page, including multiple variations of a fingerprinting function (e.g., multiple different images to be rendered through `canvas`). This allows the attacker to collect fingerprints that can be used to exploit multiple websites. Even though the credentials collected by the phishing website correspond to the specifically targeted website, in practice, these credentials might be useful for accessing other websites as well, as prior work has found that users often reuse passwords across services [26].

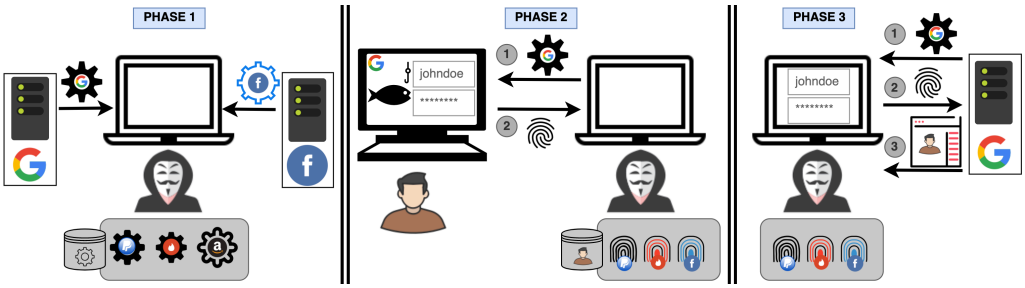

Figure 1: Overview of our attack workflow that misuses browser fingerprints for bypassing ancillary security checks.

Our attack's next phase (Figure 1 – *phase 2*) revolves around an unsuspecting user being tricked into visiting the phishing site and providing their credentials ❶. When visiting the attacker's site, the page's JavaScript code (which was extracted from the target sites) will be executed on the client side and generate the fingerprints of the victim's device ❷ that are required by the target websites for authentication (i.e., the website *knows this* device for *that* user). With the users' fingerprints and credentials at their disposal, attackers can now log into the victims' accounts. In the final attack phase (Figure 1 – *phase 3*), the attacker visits the target websites for which they have acquired the victim's fingerprints, and attempts to log into the victim's accounts ❶. At this point, the attacker spoofs their *own* device's fingerprints to mimic those of the victim's device ❷. This deceives the target websites' risk-assessment systems into considering the attacker's device as *known*. As a result, they will refrain from presenting a 2FA challenge, thus allowing the attacker to gain access to the victim's accounts ❸.

The two most crucial aspects of our attack are (i) identifying and extracting the fingerprinting code from the vulnerable websites that will be incorporated into the attacker's site, and (ii) spoofing the attacker's fingerprints to match those of the victim's device. We have built two browser extensions that streamline these tasks and eliminate the need for manual effort. In the remainder of this section we describe the two extensions and provide technical details about their implementation.

3.1 Extraction of Fingerprinting Code

Our `FP-Extractor` extension runs at "`document_start`", so that its content script is executed before any of the site's scripts. The content script injects JavaScript code directly at the top of the page (i.e., with a `<script>` tag) to be executed first. It is important to ensure that the injected code runs first, as it hooks the properties and methods that are typically used for fingerprinting, in order to keep track of accesses to these properties and the relevant API calls with their arguments. Based on the information logged, our extension generates and exports JavaScript code that corresponds to the exact fingerprinting code of the target website. Our extension currently targets the fingerprinting techniques employed by popular libraries and tools (e.g., `FingerprintJS` [5], `AmIUnique` [2], `OpenWPM` [10]).

JavaScript objects. The injected code hooks various properties of the `Navigator`, `Window` and `Screen` objects. These properties provide an abundance of information about the user's system/device, and it is a common practice for fingerprinting scripts to collect these properties. A complete list of these properties is provided in the Appendix. We note that we are not interested in the values of these properties during the code extraction process, as our goal is only to identify which properties are being used by the target website. By knowing which properties are needed, our extension generates code that will collect this information in our honeysite.

Canvas and WebGL. These two APIs are used for drawing graphics using JavaScript. They use the HTML5 `<canvas>` element; the Canvas API is typically used for drawing 2D images while WebGL for 3D graphics. This fingerprinting technique draws text and shapes on the canvas and uses `ToDataURL()` to get a Base64-encoded representation of the binary pixel data of the image [16].

`FP-Extractor` identifies when a canvas object is created by hooking the `createElement` method and checking whether "canvas" is passed as an argument when the method is called. When the canvas is created, the extension records the values of its `height`, `width` and `style.display` properties. Then, it gets the rendering context (`CanvasRenderingContext2D`, `WebGLRenderingContext`), and traces all the methods that are available to that context (i.e., `fillStyle`, `fillText`, `strokeStyle`, `font` etc). Since these methods are the ones that specify what is drawn on the canvas, our extension records all the methods called along with their arguments. Finally, it also traces the use of the `toDataURL` and `getImageData` methods that return the image data. In the case of WebGL, we also trace the invocations of the `getSupportedExtensions`, `getParameter` and `getExtension` methods that return WebGL's constants and supported extensions. With the information recorded, the extension is able to generate and export code that replicates the canvas/WebGL fingerprinting code of the target websites. In its simplest form, this code creates the canvas element, initializes the rendering context, draws the respective images on the canvas, and returns the image data. Importantly, the images that are drawn and the fingerprints that are generated on the attacker's website are identical to those generated when the victim visits the target website.

Enumeration of supported fonts. A common technique for fingerprinting fonts is to include multiple `<span>` elements in the page that contain the same text but use a different font family in each element. By measuring the dimensions (i.e., `offsetWidth` and `offsetHeight`) of each span element, the website can determine which fonts are supported by the user's system, as the dimensions of specific elements will deviate from those that correspond to fonts that are unavailable (as those will all use the same fallback font). For extracting the code that fingerprints fonts, our extension detects when the `offsetWidth` or `offsetHeight` properties of a span element are accessed, and logs information about the `textContent`, `fontSize` and `fontFamily` of the span element. In this way, the extension obtains the list of fonts that a target website tests. The extension then generates and exports code for creating the respective span elements in the attacker's website and comparing their dimensions for determining which fonts are available.

3.2 Fingerprint Spoofing

For basic fingerprints that correspond to properties of the `Navigator`, `Window` and `Screen` objects, our `FP-Spoofer` extension uses the `Object.defineProperty` method to determine when these properties are being accessed and overrides their values according to the victim's values. Our extension also compares navigator properties, deleting those that do not exist on the victim's machine and only adding those that the attacker's machine lacks. Since websites can list the properties present in the navigator object, when spoofing a property with `Object.defineProperty` we also set it to be enumerable so that it will be retrieved or looped through with `Object.keys` or the `for-in` loop.

`FP-Spoofer`'s background script changes the `User-Agent` request header to match the victim device's User Agent so that the two values are consistent, and also spoofs other headers to make the request consistent. For example, we need to spoof `Sec-CH-UA` to match the User Agent, as well as `Accept-Language` if it differs from the victim's machine.

For canvas and WebGL fingerprinting, our extension detects when the `toDataURL` and `getImageData` methods are called and returns the respective values recorded on the attacker's website. In general, for spoofing the user's fingerprints during the attack phase, our extension does not need to manipulate any intermediate values. It is sufficient for our purpose to only spoof the final values, that the website checks for determining if the fingerprints match those of the authenticating user. In the case of canvas fingerprinting we only need to return the Base64 value of the image data that was previously recorded on the attacker's website for the specific user. For WebGL fingerprinting, in addition to `toDataURL`, our extension also detects when the `getSupportedExtensions`, `getExtension` and `getParameter` methods of the WebGL API are called, and returns the WebGL extensions and constants that were recorded previously on the attacker's website for that user.

For font fingerprinting, our extension detects when the website's code accesses the `offsetWidth` or `offsetHeight` properties of span elements. If the element that is accessed corresponds to a family that is supported by the victim's browser, as recorded on the attacker's website, our extension modifies the values of `offsetWidth` and `offsetHeight` to appear different from the default ones. On the other hand, if the span element corresponds to a font that is not supported, our extension returns the baseline values for this span element. The `AudioContext` API can be used to extract a consistent fingerprint, by exploiting subtle differences in the rendering of a fixed audio waveform. Using `AudioBuffer.getChannelData()` for a generated audio snippet will return an array of floating-point values that represent the sound. We replace the array with the values from the victim's machine. Finally, while we did not find any websites that use canvas fonts for their authentication process, we describe our approach in the Appendix, for completeness.

Essentially, as the implementation of the fingerprinting techniques is **identical** on the target website and the attacker's website (since we extracted it from there), our extension knows exactly which methods are invoked and which values need to be spoofed. Furthermore, it is worth noting that while there exist browser extensions that allow users to spoof their fingerprints, such as CyDec Security Anti-Fp [13] and User-Agent Switcher [15], these are not suitable for carrying out our attack as they only support spoofing for basic fingerprints and cannot be used for advanced or non-generic techniques. A *video demonstration* of our spoofing tool's capabilities against the AmIUnique system is available here [3].

4 Experimental Evaluation

Our experiments explore the feasibility of our attack for bypassing 2FA authentication. First we identify a set of *potentially vulnerable* websites that use fingerprinting and implement 2FA, and infer whether their risk-based authentication engine leverages browser fingerprints for "*remembering*" the user's device. Then, we go through a systematic and rigorous testing process to assess whether these websites are susceptible to our attack.

Identifying potentially vulnerable websites. While our fingerprinting extraction, generation, and spoofing pipeline is fully automated (§3), identifying a set of candidate target websites and uncovering the inner workings of their risk-based authentication requires manual effort. Since a considerable amount of manual effort is required for creating accounts on different services and navigating the account settings to identify and enable 2FA, we focus our efforts on a small set that are potentially vulnerable to our attack. To that end, we first run an exploratory process that intends to identify candidate websites that run fingerprinting code on their login page, and then determine which of these websites implement 2FA. While this does not guarantee that all such websites use fingerprinting for authentication, it reduces the set of candidate websites as it excludes those that do not run any fingerprinting code.

Discovery of login pages. The first step in this process is to identify websites that support account registration, and to locate their login pages. We follow the methodology of Drakonakis et al. [31] for detecting pages that include login or registration forms. If no login forms are detected on the landing page, our crawler follows all the links on the landing page that point to pages under the same domain, and analyzes the pages' URLs for the presence of indicative keywords (e.g., login, sign in etc.). With this approach we have located the login pages of 11,527 websites from the Alexa top 20K websites (5,736 and 5,791 from the top 10K and the top 10K-20K websites, respectively).

Fingerprinting detection. We use the Chrome browser with `FP-Extractor` installed, and visit the landing and the login pages of the websites that we have detected during the discovery process. When visiting a page our extension logs all fingerprinting calls. At this point we do not need our extension's code extraction functionality, but are only interested in logging which fingerprinting calls are invoked; this information is sufficient for determining if a website uses fingerprinting.

Determining 2FA support. The next step is to identify the subset of websites that implement 2FA, out of those that use fingerprinting on their login pages. For this, we first search for relevant terms, such as "multi", "factor", "authenticator", "remember device" etc., in the login pages' source code using regular expressions. We also expand our set of websites to include websites that we know through personal use that they support 2FA. We manually inspect the websites' source code to identify which scripts are responsible for authentication, 2FA and fingerprinting. While using our extension helped us create an initial set of candidates for our experiment, this does not provide information about the techniques used at a script-level granularity. Our manual analysis reveals that websites commonly include multiple scripts that implement fingerprinting, and our extension-based approach cannot differentiate which functionality was the result of each script. For a more fine-grained analysis we use `VisibleV8` [42] on the login pages of the websites that use fingerprinting, which allows us to log all native functions and property accesses during JavaScript execution, at the granularity of individual scripts. This process provides useful contextual information for our analysis.

4.1 Experimental Methodology

In the previous step we described our process for identifying websites that are potentially vulnerable to our attack. Here we describe our methodology for testing the candidate websites in order to determine (i) whether they use fingerprints for authentication and (ii) if they are indeed susceptible to our attack. Overall we tested 300 websites; our findings are presented in §4.2.

Account registration. To be able to test these websites, we first need to register an account on them and manually log into these accounts to enable 2FA if there is such an option available in the settings. We also provide a valid phone number during the account registration or when 2FA is enabled.

Testing devices. For our experiments we use two devices with different operating systems and browsers. Our primary device is a MacOS laptop running Chrome (version 85.0.4183.83), and our secondary device is a Windows laptop running Edge (version 85.0.564.44). The primary device plays the role of the victim's device, and is the device used for registering the accounts and enabling 2FA. For websites where we test existing personal accounts, this is the device that has been used to access these accounts in the past. In general, our primary device is the one that these websites *remember* and consider as *known*. On the other hand, the secondary device, which represents the attacker's device, has never been used to access these accounts in the past. We expect websites that use 2FA to consider this device as *new* and, therefore, to trigger a 2FA challenge when the user logs in using this device. Furthermore, to avoid "polluting" subsequent experiments from the secondary device, we never solve a 2FA challenge when presented, so websites will not consider this device as known in any future attempts since the authentication process fails.

Testing procedure. We follow a systematic approach for testing the candidate websites. We have devised a series of specific steps to be followed, and rely on differential testing to understand how each website's authentication system behaves, how 2FA is triggered, and how our attack can bypass 2FA. More specifically, our methodology tests if a website (i) uses 2FA during authentication, (ii) uses cookies or fingerprints to remember devices, and (iii) imposes restrictions based on the device's IP address. Furthermore, we follow this procedure twice, once for testing each website's default settings, and once after explicitly enabling 2FA, if such an option is available.

For every website to be tested, we first log into our account using our primary device and select the "*remember this device*" option, if available. Then we logout and re-login, to check if the website does indeed *remember* the device (i.e., does not present a 2FA challenge). At this point we also log into the website using our secondary device, which has never been used to access this account in the past, and check if a 2FA challenge is presented. Since the secondary device is not known to the website, we expect 2FA to be triggered in this case. For websites that present a 2FA challenge when using the secondary device, while at the same time they appear to remember the primary one, we explore whether this happens due to the use of fingerprinting. Specifically, we clear all browsing data (e.g., cookies, local storage) on the primary device and log into the account again. If the website still remembers the device, this is an indication that it uses fingerprinting to determine if the device is known.

Bypassing 2FA. For the websites that use fingerprinting to remember the user's device and trigger 2FA when logging in from a new device, we use our extension to test if they are susceptible to our attack. We first visit the login page of target websites using a browser with `FP-Extractor` to export JavaScript code that generates the same fingerprint as the target website. Then, we mount this code in our honeysite (i.e.,

attacker's website) and visit the honeysite with our primary device (i.e., victim device) to obtain the device's fingerprint. After acquiring the fingerprint of the victim's device we log into the target website using the secondary device, where `FP-Spoofer` will modify the device's fingerprints to match those of the primary device. Our attack is deemed *successful* if the secondary device *does not* receive a 2FA challenge.

IP address/Geolocation. We observe that certain websites consider the device's IP address/geolocation information as a signal for determining whether 2FA should be triggered or not. Specifically these websites check whether the IP address or IP-based geolocation information (e.g., country, city etc.) of the device that is currently logging into the account matches those from previous user logins. Depending on how restrictive this check is, it can raise the bar for the attacker or even prevent our attack; checks that require IP addresses to match the user's country or city can be easily bypassed using proxies and VPN services. Onaolapo et al. [55] found that attackers actually employ such strategies in the wild. However, websites that only accept IP addresses that have been used by the user before can pose a significant challenge to the attacker. During our experiments, we systematically assess this aspect of the authentication process and use a VPN to test IP addresses from different ISPs, cities, and countries. We have also devised a technique that attempts to bypass such IP-based restrictions; we modify our network requests when running our attack and include the victim's IP address (collected when they visited the phishing page) in an `X-Forwarded-For` header. This header is typically used for specifying the originating IP address when traffic goes through a proxy [8]. This allows the attacker to pretend that they are actually behind the victim's IP address and are using a proxy when attempting to log into the user's account.

Inferring fingerprinting-based authentication checks. In practice, the attacker does not need to know which fingerprinting attributes collected by a target website are actually used for the authentication process, as our attack pipeline extracts and replicates all fingerprinting techniques. However, for our analysis we are interested in obtaining a more fine-grained and in-depth understanding of risk-based authentication systems that use fingerprints. As such, for websites that are vulnerable to our attack, we systematically evaluate whether each fingerprinting vector actually affects the authentication process. Due to the prohibitively large number of potential combinations, we follow a strategy based on the *process of elimination*. In more detail, we repeat our attack multiple times, where in each attempt we remove one of the fingerprinting attributes contained in the user's fingerprint profile. Depending on whether each attack instance results in 2FA being triggered or not, we can infer whether that specific fingerprinting vector is part of the risk-based authentication checks. By repeating this process for all the fingerprints collected by that web service, we can identify the *absolutely minimum* set of fingerprints required to manipulate the authentication process. It is important to note that we repeat our experiments numerous times over the course of multiple months, to ensure the validity of our findings and avoid false positive (i.e., labeling an attribute as necessary even though it is not) due to some other mechanism being triggered (e.g., multiple consecutive logins triggering a rate-limiting mechanism).

Table 1: Fingerprinting attributes used by websites with a detectable login page (within the Alexa Top-20K).

Technique	Top 10K		Top 10K-20K	
	Home	Login	Home	Login
Navigator	5,510	5,403	5,587	5,371
Window	5,261	5,104	5,272	4,968
Screen	5,209	4,682	5,231	4,473
Timezone	5,035	4,617	4,934	4,282
Canvas	1,224	1,254	1,077	879
Canvas Fonts	179	380	142	237
WebRTC	221	313	192	210
AudioContext	290	351	223	234

4.2 Experimental Results

Here we present our experiments exploring the feasibility and effectiveness of our attack in the wild.

First, in Table 1 we provide statistics on the prevalence of fingerprinting techniques for websites in the Alexa top 20K for which we were able to identify their login page (i.e., 5,736 and 5,791 websites in the top 10K and top 10K-20K datasets, respectively). We observe that the majority of websites, in both datasets, collect basic fingerprints. Furthermore, we observe a clear trend of websites in the top 10K dataset employing more advanced fingerprinting techniques on the login pages compared to their home page. The websites in the 10k-20k dataset exhibit a more uniform deployment of advanced techniques. Notably, while we observe widespread deployment of fingerprinting vectors, these are not often incorporated into the websites' authentication process, as we will detail next. We hypothesize that these are more likely used for tracking purposes.

We select a subset of 300 popular websites from our discovery process that implement fingerprinting and support 2FA for manual exploration and testing. These were selected based on our experiment on websites with fingerprinting code on their login pages and being listed on [1]. Our experiments reveal that only 16 out of the 300 websites use fingerprints for *remembering* the user's device, while the rest rely on browser cookies for this. Interestingly, the tested websites included four banking and eight tax-preparation websites, of which two and four respectively use fingerprinting for authentication. As such, our experiments indicate that (i) high-value and financial services tend to adopt security mechanisms such as 2FA in order to better protect their users' accounts, and (ii) augmenting the authentication process with fingerprints is disproportionately used among such high-value services, signifying the severe ramifications of our attack. As 2FA becomes more prevalent [14], we expect that risk-based authentication that uses fingerprints will also become more common.

Table 2: Risk-based authentication mechanisms in popular web services we evaluated against our attack. For IP address restrictions we explicitly note if using the `X-Forwarded-For` header (↦) or IP addresses from the same city (⊗) is effective.

Website	Fingerprinting Technique				IP Address Restrictions		Vulnerable
	Basic FP	Canvas/WebGL	Fonts	Audio	IP Check	Bypass	
Bank-A	✓	✗	✗	✗	✗	-	✓
Bank-B	✗	✗	✗	✗	✓	✗	✗
CreditCard	✓	✗	✗	✗	✓	↦	✓
Trading-A	✓	✗	✗	✗	✗	-	✓
Trading-B	✗	✗	✗	✗	✓	↦	✓
Tax-A	✓	✓	✗	✗	✓	✗	✗
Tax-B	✓	✓	✓	✗	✗	-	✓
Tax-C	✓	✓	✓	✓	✗	-	✓
Tax-D	✓	✓	✓	✓	✓	✗	✗
eCommerce-A	✓	✓	✗	✗	✗	-	✓
eCommerce-B	✓	✗	✗	✗	✓	✗	✗
RideSharing	✓	✓	✓	✗	✓	↦	✓
Food&Beverage-A	✓	✗	✗	✗	✓	⊗	✓
Food&Beverage-B	✓	✗	✗	✗	✓	✗	✗
AdBlocking	✓	✗	✗	✗	✓	⊗	✓
WebInfrastructure	✓	✗	✗	✗	✓	✗	✗

Table 2 details the findings from our attack evaluation on the authentication mechanisms of these 16 websites. Our main focus here is on the first 14 services which trigger 2FA when a new device is used to access an account. We also include two services (`WebInfrastructure`, `AdBlocking`) that highlight additional dimensions of risk-based authentication. Due to the severity of our attacks and the fact that accounts on certain services are extremely valuable and highly-targeted, we present them in an anonymized form that denotes their category.

As can be seen in Table 2, our attack can successfully bypass 2FA in 9 out of the 14 websites. The five websites that are not vulnerable to our attack require the device's IP address to match one of the IP addresses previously used to access that account, and are not deceived by our `X-Forwarded-For` ploy. Furthermore, we found that 12 of the 14 websites use fingerprinting to determine if the authenticating device is known and whether 2FA should be triggered or not. From the total of 14 websites that use fingerprinting, eight rely on basic fingerprints (e.g., properties of `navigator`, `window`, etc.), and six of the tested websites use more advanced fingerprints for authentication like canvas/WebGL and fonts. Finally, two of those also use audio fingerprinting for the purpose of authentication. We provide additional details for interesting use cases in the Appendix.

Our exploration revealed another dimension of the use of fingerprinting for authentication. Basic fingerprints remain the same across different sites, since they correspond to the user's system characteristics and properties and do not change as long as the environment remains the same. For advanced fingerprinting techniques, however, a site is able to make its users generate fingerprints that are different from those generated when visiting other sites (e.g., by rendering a unique image). While this may prevent the attacker from creating "generic" fingerprints that can be used on multiple websites, *our attack is still effective* since we extract the fingerprinting code from each target website and generate appropriate site-specific fingerprints.

Our analysis shows that only six of the 16 websites employ advanced fingerprinting techniques. To make matters worse, three of the tax-related websites use the *default* implementation of `Fingerprintjs2` for the advanced fingerprints. This results in these three websites rendering the same images for canvas fingerprinting and loading the same list of fonts for fonts enumeration. We also found that `Tax-C` and `Tax-D` use the same audio snippet for audio fingerprinting (`Tax-B` employs an earlier version of `Fingerprintjs2` that does not support audio fingerprinting). As a result, an attacker who uses `Fingerprintjs2` (or the code extracted from one of these websites) can obtain the fingerprinting values required for bypassing 2FA in all three websites. To further explore this issue, we visited the websites of 10 additional popular banks to check whether they use advanced fingerprinting techniques, and observed significant overlap across the images and fonts lists they use. Even though each website renders between 1-5 images, there are only nine images and two random images in total across the ten websites. Two websites with JS font fingerprinting use identical font lists, and five websites with canvas font fingerprinting use one of two font lists. The two sites with audio fingerprinting render the same audio wave form.

IP constraints. While `Bank-B` and `Trading-B` do not use any JavaScript-based fingerprinting attributes but only rely on the `UserAgent` HTTP header, which can be trivially spoofed, we include them in our analysis to illustrate the challenge posed by IP address checks as well as the dangers of trusting `X-Forwarded-For`. In more detail, regarding IP-based constraints, we found that only 11 websites perform such checks for determining if the login attempt is suspicious. Our attack can bypass the IP address restrictions in three websites using the `X-Forwarded-For` header in outgoing requests (`CreditCard`, `Trading-B`, `RideSharing`). Moreover, we found two websites that do not trigger 2FA if the authenticating IP address matches the user's city. With the wide availability

of VPN and proxy services, we consider such coarse-grained checks to be inadequate for protecting valuable accounts.

Cookie hijacking. Up to this point we have focused on attackers that have the account credentials (e.g., obtained through phishing), as that is the most common account hijacking vector according to prior research [23]. Nonetheless, recent work by Drakonakis et al. [31] demonstrated the feasibility of cookie-hijacking attacks at scale. More importantly, the authors noted the lack of additional fraud-detection checks (which occur during the log in process) when attackers use stolen cookies as those belong to sessions that have already been "validated". In fact, the authors found only one instance where they were not able to access the victim's account due to such checks. To that end, we include `WebInfrastructure` to test whether our attack can also be leveraged by cookie hijackers. For this experiment we visit `WebInfrastructure` using our primary device, log into our account, and export `WebInfrastructure`'s cookies from the browser. Then, we import these cookies into a different browser on our secondary device, and visit `WebInfrastructure`. Initially we found that we were, indeed, unable to access our account using the "stolen" cookies. Upon a more in-depth analysis we found that the site uses the device's User Agent (obtained through JavaScript and the HTTP header) to detect suspicious logins, but also checks the IP address. As such, when attempting to access the account using the stolen cookies *and* `FP-Spoofer`, we could only gain access if the secondary device's IP address was one previously used by the victim. As such, while fingerprint-spoofing allowed us to pass the corresponding checks, the IP address check effectively prevents the attack.

Email alerts. While our experiments focus on bypassing 2FA, we include our analysis of `AdBlocking`, as it highlights an additional dimension of risk-based authentication. In more detail, `AdBlocking` accounts have 2FA disabled by default, but the service alerts users about successful logins that occur from new devices or from IP addresses that are not from the same city. However, we found that by spoofing the fingerprints we can trick the service into not sending the email alert.

Behavioral evolution over time. We re-tested affected services at least 20 times over a period of six months (04/2021 - 09/2021), even after our disclosure. Interestingly, we observed cases where the risk-based authentication behavior changed over time. In our initial experiments with `Trading-A`, their system required the user to solve a 2FA challenge every time, if 2FA was explicitly enabled by the user. For the default setting, however, the system used basic fingerprints to determine if a 2FA challenge should be presented. When re-analyzing `Trading-A` some time after our disclosure, we observed that `Trading-A` now requires the user to provide a phone number when registering a new account and that 2FA is enabled by default for new accounts. We cannot tell, however, if this change happened organically, or due to our disclosure and recommendations. Surprisingly, 2FA has not been retroactively enabled for existing accounts, resulting in different default levels of protection for accounts. Another example is `Tax-B`; in our initial experiments it used fingerprinting to remember the device but now relies on the presence of cookies to determine if the device is *known*. Again, we cannot tell if this change was the result of our recommendations. Our final example is that of `WebInfrastructure`, which matched the device's IP address and User Agent in our initial experiments. However, prior to our disclosure we observed that these checks were removed and we can now successfully access the account using "stolen" cookies, regardless of the device or IP address we connect from. We do not know what led to the removal of these additional security checks, but have disclosed our findings to them and they are currently investigating the issue.

Table 3: Phishing website datasets. JS denotes the websites for which we have JavaScript execution traces, and FP denotes the phishing sites that collect browser fingerprints.

Dataset	Time Period	Sites	JS	FP
`Phish-A`	31/05/2018 - 19/06/2019	71,343	39,618	29,312
`Phish-B`	31/10/2018 - 05/05/2020	82,431	40,777	36,733
`APWG`	05/05/2020 - 12/04/2021	173,269	93,568	85,491

5 Phishing and Fingerprinting

In this section we focus on the phishing ecosystem and present a large-scale exploration of the phishing sites obtaining users' browser fingerprints. We correlate the information that phishers currently collect with the findings from our empirical analysis in §4 to assess whether attackers are already collecting sufficient fingerprinting attributes for carrying out our attack.

Datasets. Table 3 details the datasets used for our analysis. We obtained the two datasets (`Phish-A`, `Phish-B`) from the authors of [76], which include more than 153K phishing sites that appeared over a two-year period. They also include the corresponding JavaScript for 80,395 of the sites. While `Phish-B` does not include labels for the target website (e.g., if the phishing site is masquerading as Paypal) we cross-referenced that dataset with information made available by the Anti-Phishing Working Group's (APWG) eCrime Exchange (eCX) repository, allowing us to obtain the missing labels. Finally, we obtained the `APWG` dataset directly from the eCX repository [12], which provides a more recent and extensive snapshot of the phishing ecosystem over an 11-month period. Together, these datasets provide a broad and representative view of the phishing ecosystem over a three year period.

JavaScript execution traces. To better understand if phishing websites are using fingerprinting and whether they are collecting fingerprints that would allow the attacker to carry out our attack, we use `VisibleV8` to dynamically analyze the JavaScript code of the phishing sites in our datasets. For the `Phish-A` and `Phish-B` datasets we were provided with the HTML files of the phishing websites as well as the JavaScript

Table 4: Phishing sites that implement fingerprinting.

Technique	Phishing Datasets		
	Phish-A	Phish-B	APWG
Navigator	27,578	34,650	84,239
Window	24,848	23,650	73,258
Screen	10,244	26,856	57,633
Timezone	22,636	28,549	59,251
Canvas	3,508	5,395	11,650
Canvas Fonts	56	91	399
WebRTC	536	165	1,938
AudioContext	275	363	1,795

Table 5: Phishing sites that obtain all the necessary browser fingerprints for bypassing 2FA in the target sites. "*" indicates a mismatch in fingerprinting function arguments.

Target	Phish-A		Phish-B		APWG	
	Sites	Bypass	Sites	Bypass	Sites	Bypass
Bank-A	83	1	685	14	330	74
Bank-B	1,549	-	2,683	-	327	-
CreditCard	89	61	0	0	12	0
Trading-A	0	0	0	0	6	6
RideSharing	7	0	363	1*	1378	5*
WebInfrastructure	0	0	1	1	220	219

files they load. To that end, we deploy them on our own local server and re-write the origin URL of the JavaScript files loaded to point to the corresponding JavaScript files in our datasets. This allowed us to analyze phishing sites that are not available anymore due to the sites being taken down or the original versions of their JavaScript files not being available anymore. For the more recent APWG dataset we visit the actual phishing sites, as this dataset does not include a snapshot of their code. We use VisibleV8 to load each phishing site and log all the JavaScript calls along with their arguments. To ensure that websites' JavaScript code is executed, we interact with the pages in an automated way to emulate simple user behavior (e.g., scrolling, making random mouse movements and clicks). Based on the JavaScript execution traces that we extract from VisibleV8's logs we determine which fingerprinting techniques each phishing website implements and which attributes are collected.

Table 3 presents the phishing datasets we used in our analysis, the number of websites in each dataset that run JavaScript, and how many of them are collecting fingerprints. A general observation is that the percentage of phishing websites that *appear* to run JavaScript is lower than what we would have expected, across all 3 datasets (i.e., between 49.46% and 55.53%). We manually checked 25 random phishing sites from the APWG dataset that did not produce a JavaScript execution trace (recall that for this dataset we visited the actual phishing sites) and found that 14 and three return a 404 and 403 error respectively, while three other sites show a static page with an "account suspended" message. From the remaining websites one is a shortened URL that has been flagged by Bitly as potentially harmful, and another uses a shortened URL for a Google Forms site, but has been suspended by Google for violating their terms of service. Finally, one site has no content, one includes an empty local JavaScript file, and one shows a popup window asking for a username and password. As such, apart from the unavailability of resources or sites being taken down, we believe that client-side cloaking techniques [76] have likely affected the collection of JavaScript across all three datasets. Interestingly, for the phishing websites with JavaScript execution traces, we find that the majority collect fingerprints, with 73.98%, 90.08% and 91.36% across the 3 datasets respectively. We also observe an increase in the number of websites collecting browser fingerprints over time.

Table 4 presents the number of phishing sites that collect various types of fingerprints. The study by Zhang et al. [76] on phishing sites' cloaking strategies reported checks for simple browser attributes (specifically, the User Agent and whether cookies are disabled) on approximately 23% of the phishing sites. Our analysis provides a more comprehensive picture of the three datasets as we detect all common fingerprinting techniques, while also revealing the widespread use of advanced fingerprinting techniques across the phishing ecosystem. Specifically, we find that in total 28,526, 35,653 and 85,461 (i.e., 39.98%, 43.25% and 49.32%) websites from the three datasets collect basic fingerprints with the majority of them being properties of the Navigator, and that between 5% and 7% collect advanced fingerprints, with canvas fingerprinting being the most prevalent technique among them. We also explore whether and how phishing sites send fingerprinting values back to their servers; we provide more details in Appendix E.

Bypassing 2FA. Next, we analyze the subset of phishing sites that target one of the services from Table 2 and for which we have their JavaScript execution traces. Specifically, we cross-reference the fingerprinting attributes that each phishing site collects with those necessary for manipulating the target's risk-based authentication mechanisms to bypass 2FA. As can be seen in Table 5, Bank-B is an extremely popular target for phishing websites. Since Bank-B only relies on the User Agent HTTP header and does not check navigator.userAgent, essentially every phishing site has sufficient information to pass the device-based check. Nonetheless we include it here as a point of comparison. Additionally, since we do not have historical information of when the IP-address-based check was deployed by Bank-B we cannot conclude how many phishing sites would have been able to bypass 2FA in practice.

On the other hand, we find that Bank-A is not only a popular target, but that the number of phishing sites that collect the appropriate fingerprints is significantly larger in our most recent dataset; while 8.1% of the phishing sites are capable of bypassing 2FA in Bank-A across all datasets, in the most recent dataset the ratio climbs to 22.42%. This indicates that phishers are adapting over time to the risk-based mechanisms employed by high-value websites like Bank-A. Interestingly, while we find that six phishing sites collect all the necessary fingerprinting attributes used by RideSharing, the actual arguments passed to two dynamic fingerprinting functions

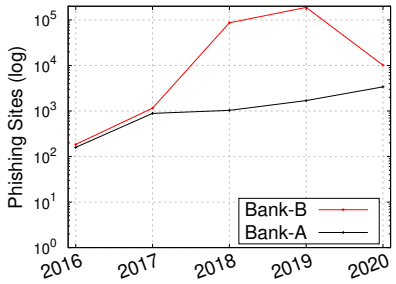

Figure 2: Phishing websites targeting `Bank-A` and `Bank-B`.

(for canvas and font fingerprinting) are not the same as those used by `RideSharing`, thus rendering their overall fingerprint a mismatch. Further inspection reveals that in five cases the mismatch is due to the phishing sites using the default values used by popular fingerprinting libraries, while the final instance is using a library by an anti-bot service.

Longitudinal trends. To get a broader view of phishing sites potentially adapting their targets over time, we use APWG's eCX service to obtain the phishing domains that targeted `Bank-A` and `Bank-B` between 2016-2020, as shown in Figure 2. While in 2016 and 2017 the two banks were targeted by a comparable number of phishing sites, `Bank-B` phishing sites aggressively increased in 2018 and continued to increase in 2019, but had a sharp decline in 2020. On the other hand, the number of phishing sites targeting `Bank-A` steadily increase from year to year. While we cannot conclusively infer the root cause of this trend without detailed knowledge of the risk-based authentication checks the banks enforced throughout this entire period, Figure 2 and Table 5 indicate that phishing sites may be adapting their targets based on the obstacles presented by risk-based authentication. In other words, since bypassing 2FA in `Bank-A` currently only requires spoofing certain fingerprints while `Bank-B` also requires exactly matching the user's IP address, `Bank-A` presents a more attractive target to phishers.

Summary. Overall, while our analysis is limited to phishing sites for which we were able to obtain their JavaScript code, our findings show that phishing sites are not yet widely replicating the fingerprint-generation process of targeted websites. Nonetheless, the cases of `Bank-A` and `CreditCard` highlight the risks that users face and the need to improve existing risk-based authentication deployments, as we discuss in §6.

6 Discussion, Limitations and Defenses

In this section we further discuss our experiments and findings.

Ethics and disclosure. The severity of our attack necessitates the responsible disclosure of our findings to the affected vendors. As such, we disclosed our methodology and findings to every vulnerable website through their bug bounty programs or security contact points when those were available. When we could not find contact points dedicated to security issues we reached out over their general contact email address. At the time of writing, six vendors have responded. `Bank-A`, `Tax-A` and `WebInfrastructure` requested additional details and proof-of-concept demonstrations, which we provided. `Bank-A`, subsequently, verified our attack and is currently working towards a fix. `eCommerce-A` informed us that they were aware of the issue. It is important to emphasize that all of our experiments were conducted using test accounts or our personal accounts. We did not interact with or affect other users, nor did we collect browser fingerprints from any users.

Fingerprinting and authentication prevalence. While using fingerprints for augmenting the authentication process is not a new concept [17], our experiments reveal that this has yet to become widespread practice. However, as fingerprinting has gained significant traction in recent years, and third-party libraries have started supporting the use of fingerprints for authentication (e.g., [6, 9]), it is likely that such mechanisms will become far more common in the near future. Additionally, our research shows that while in many cases fingerprints may be used for augmenting the authentication process, other signals carry more "weight" (e.g., the presence of cookies and the device's IP address). Unfortunately, our experiments show that *high-value services (e.g., banks, tax services) are most commonly vulnerable to our attack*. As such, while the attack that we demonstrate may not yet be a widespread threat, the severity of the affected web services and the overall implications of their user accounts being compromised, highlight the need for alerting developers about the security implications of leveraging device fingerprints for the authentication process. We also hope that our work kickstarts a wider discussion within the research community and incentivizes additional research on fingerprinting schemes that are robust to spoofing.

Attributes. Our extensions target the fingerprinting vectors used by popular libraries and websites. If a website uses custom techniques or those libraries incorporate additional techniques, our extensions would need to be expanded for handling them.

2FA mechanisms. While SMS is the most commonly deployed 2FA technique, despite its well-documented shortcomings [29,41,51], our attack is not limited to a specific 2FA mechanism but instead provides a method for deceiving the risk-assessment engine that decides whether a 2FA challenge should be triggered. For instance, `eCommerce-A` supports the use of authenticator apps for 2FA, and our attack bypasses that as well.

MITM phishing toolkits like Evilginx [4] allow attackers to deploy phishing websites with man-in-the-middle capabilities for using phished credentials to log into target websites in real time (i.e., when the victim is interacting with the phishing site) and then also trick the victims into divulging a 2FA code, thus allowing the attacker to log into the victim account. However, the major limitation of this attack is that in high-value services that only use short-lived session cookies the attacker can only access the victim's account that one time and would fail in future attempts due to the 2FA challenges.

Guidelines for vendors. As we demonstrate, certain techniques for augmenting authentication may actually undermine the overall security posture of a given service.

Two-factor authentication. In our experiments we found that only 8 of the vulnerable services we have identified offer an option to mandate that a 2FA challenge has to be passed for every login attempt (one more site offers that option only for transactions). Moreover, in all cases, that option is optional and users have to explicitly enable it. As such, we argue that all websites should provide such options, as that would allow users to fortify their account against our attack, while also significantly raising the bar for attackers in general. We believe that this option should be `opt-out` instead of `opt-in`, especially in high-value or highly-sensitive services, to further nudge users towards improving their security hygiene; indicatively, Google recently automatically enrolled 150 million users in 2FA [7]. We do note, however, the friction that additional authentication requests and factors can cause. This tradeoff between usability and security has been studied extensively [27], and recent reports found that users are in favor of strengthening security in high-value services through 2FA [59]. Finally, while this is not pertinent to our attack, since it is not affected by the actual form of 2FA mechanism selected by the user, vendors should strive to adopt and offer more secure 2FA options (e.g., U2F, authenticator apps).

Risk signals. Our experiments revealed that certain vendors incorporate fingerprints into the authentication process, but other signals play an important role and can affect the feasibility of our attack. We have identified two signals that vendors should use for identifying suspicious logins and triggering 2FA. First, we found that certain sites will always trigger 2FA if the request doesn't include certain HTTP cookies. While there are legitimate scenarios where this occurs (e.g., the user has cleared the browser's browsing data), it can also indicate that the login attempt is from a new/unknown device. Obviously, using this signal would not be effective against cookie hijackers. Second, we found that certain sites have more stringent IP-based checks. While attackers can easily use proxies or VPN services to "obtain" an IP address with a similar geolocation to the victim [55] (e.g., same city) stricter IP requirements (e.g., belonging to the same ISP or having been used to access that account before) present additional obstacles to attackers. Overall, as noted by OWASP [56], when alternative defenses are "implemented in a layered approach, they can provide a reasonable degree of protection". As such, a *careful* use of browser fingerprints in conjunction with other signals like IP address checking and mandating the presence of specific cookies, can lead to a more robust authentication process.

Best practices for users. Our main threat model assumes that the attacker knows the user's password. As such, the attack can be mitigated by "best practices" commonly highlighted in guides for safer Internet browsing, such as the use of password managers. Additionally, users should enable 2FA in sites that support it, and further enable options that require solving a 2FA challenge in *every* login if such an option is available (e.g., `Tax-A` offers this). Finally, users can adopt tools or browsers that affect browser fingerprinting, which we discuss next.

Anti-fingerprinting defenses. Our attack relies on our ability to accurately obtain and replicate, the user's browser fingerprints. As such, defenses [21] offered by browser extensions or privacy-oriented browsers that alter the user's fingerprints can potentially mitigate or prevent our attacks. However, this depends on the specific fingerprinting attributes covered by each defense and whether they are used by a given website. We also note that such defenses may affect or break websites' functionality. In our experiments we also visited target websites using Brave as our primary browser, which randomizes canvas fingerprints for tracking prevention, and observed that sites that use canvas fingerprinting for authentication always prompted us to solve 2FA.

Future directions. Recent research proposed using fingerprints to augment authentication [44] by "chaining" sessions, with a random canvas fingerprint being generated in each session and used for verification in the following session. While this approach can effectively mitigate the phishing attack we present, it is vulnerable to other attacks. Nonetheless, we consider this an important proposal and hope that our work further incentivizes additional research in the area. While an ideal countermeasure would remove the need for chaining sessions, any approach that does not rely on memory of prior sessions must solve an inherent challenge: generating a fingerprint in a manner that cannot be spoofed. Since this is a client-side process, such an approach would necessitate leveraging some form of Trusted Execution Environment (e.g., a system like TrustJS [37]). We consider this an interesting future direction.

7 Related Work

To the best of our knowledge, this paper presents the first comprehensive security analysis of *real* risk-based authentication systems that leverage browser fingerprints, and the first demonstration of a practical attack for bypassing 2FA. Here we discuss prior work and studies around data breaches, account hijacking and authentication-augmenting mechanisms.

Van Acker et al. [70] conducted a large scale study on the security of login pages, by evaluating the presence of mixed content and the use of mechanisms like HSTS, HPKP and SRI. To detect breaches in popular services, DeBlasio et al. [28] proposed an approach that leverages honey accounts and password reuse as a method for detecting sites being compromised. Prior work has also proposed strategies for deploying risk-based authentication systems, or have studied certain characteristics of real-world deployments. In an earlier study, Hurkala and Hurkala [39] proposed a system that relies on the IP address, device profiling (i.e., User-Agent and Accept-Language in HTTP headers), presence of cookies, access time and failed login attempts. Freeman et al. [35] used a real-user dataset of login attempts from Linkedin, and classified them into benign and suspicious based on the IP address and User Agent. Steinegger et al. [67] implemented an authentication system that checks the browser fingerprint (calculated using the

`Fingerprintjs2` library), geolocation (i.e., country) based on the IP address, and the number of failed login attempts. Alaca and van Oorschot [17] identified several fingerprinting vectors that can be used for authentication and classified them based on the distinguishability they provide and their resistance to spoofing. Spooren et al. [65] explored the effectiveness of mobile fingerprints for risk-based authentication and found that they are considerably less unique than the fingerprints of personal computers. In a different line of work, Bonneau et al. [22] explored the privacy concerns that arise due to the permanence and simulatability of such features when used for authentication.

Wiefling et al. [74] explored the authentication systems of eight popular services to identify which features contribute to the computation of the risk score. To that end, they created a number of personas and corresponding accounts, and built a framework that uses virtual machines and emulates user activities. However, their experiments only focus on the User Agent string, language, and screen resolution, and as such do not provide in-depth or detailed insights on how browser fingerprints are actually being used for risk analysis in the wild. In a subsequent work [72], they performed a study with 780 users, in which they collected 247 fingerprinting features during login and assessed their suitability for risk-based authentication. In [73], they explored users' perceptions on the usability and security of risk-based authentication, and in [75] they assessed link-based and code-based re-authentication.

Previous work [19, 20] has focused on identifying fingerprinting attributes that are suitable for authentication (e.g., with high entropy, low usability cost, stability). In [20] Andriamilanto et al. proposed `FPSelect`, a tool for selecting fingerprinting attributes for authentication systems that satisfy a service's security requirements while minimizing the usability cost. In a follow up work [19], they conducted a large-scale study on the properties of browser fingerprints for authentication. They found that at least 90% of the inspected fingerprinting attributes are stable (i.e., identical values for almost six months) and can be used for authentication. In [18], Andriamilanto and Allard present `BrFAST`, a framework that incorporates `FPSelect`, for the selection of fingerprinting attributes. These studies assumed an attacker with knowledge of the distribution of fingerprints who performs a dictionary-style attack by submitting common fingerprinting values. We explore an entirely different attack where the attacker extracts a user's *exact* fingerprints and spoofs their own device's fingerprints to match these values when impersonating the user and, importantly, demonstrate the implications of this attack in the wild. Moreover, our attack can spoof all the attributes these studies proposed for augmenting authentication.

In an independent concurrent study, Liu et al. [50] explore a similar attack and demonstrate that users' fingerprints can be spoofed by an attacker. However, their approach does not detect which fingerprinting attributes are needed for different target websites, nor does it provide a method for automatically extracting and generating per-target-website fingerprinting code. It also overlooks various advanced attributes handled by our system (e.g., canvas fonts). More importantly, their attack requires significant manual effort for several attributes, including pausing execution with breakpoints and manually changing object values through the browser's debugging tools, and changing browser and operating system settings. Finally, this study does not present an in-depth exploration and evaluation of the attack against risk-based authentication systems in the wild.

Furthermore, Campobasso and Allodi [24] recently reported on an underground marketplace that sells resources for bypassing risk-based authentication. This marketplace relies on malware that infects victims computers for collecting a vast amount of information, which includes browser fingerprints for some users. The paper provides an interesting economic analysis on the impact of various resources on the pricing of account profiles. However the authors did not analyze any profiles/resources or the software used for obtaining and generating those resources, which could provide additional insight on the exact nature of the fingerprints being collected, the services being targeted, and the actual effectiveness of the attack resources offered by the marketplace for bypassing risk-based authentication mechanisms. As such, this study is complementary to our work as it indicates that attackers are indeed exploring techniques for impersonating user devices.

8 Conclusions

Critical and high-value web services have introduced additional security mechanisms and checks into their authentication workflows to prevent attackers from compromising accounts even if they are able to obtain or guess users' credentials. We presented the first empirical analysis of the *operation* and *effectiveness* of such systems in *real-world* high-value web services. Accordingly, we demonstrated how attackers can automatically extract and misuse users' browser fingerprints for deceiving risk-based authentication systems into trusting the attacker's device and bypass two-factor authentication. Our real-world experiments highlight the severity of our attack, as we show that major financial services and e-commerce services are vulnerable. We also found major services being targeted by phishers that obtain sufficient fingerprinting attributes to completely bypass 2FA. As such, our research highlights the danger of incorporating additional security mechanisms without first conducting a comprehensive and in-depth assessment of potential pitfalls. To get the remediation process under way, we have notified the affected vendors and proposed guidelines for a more robust authentication process.

Acknowledgements: We thank the anonymous reviewers, and our shepherd Mohammad Mannan, for their valuable feedback. This work was supported by the National Science Foundation (CNS-1934597). Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government.

References

[1] 2FA Directory. https://2fa.directory/.

[2] AmIUnique. https://amiunique.org/.

[3] Demo of our spoofing capabilities against the AmIUnique fingerprinting system. https://vimeo.com/629397823/b509389d0e.

[4] Evilginx - Man-in-the-middle attack framework. https://github.com/kgretzky/evilginx2.

[5] FingerprintJS. https://github.com/fingerprintjs/fingerprintjs.

[6] ForgeRock - Implementing Device Fingerprints With Intelligent Authentication Trees in AM. https://developer.forgerock.com/docs/platform/how-tos/implementing-device-fingerprints-intelligent-authentication-trees-am.

[7] Google blog - Making you safer with 2SV. https://blog.google/technology/safety-security/reducing-account-hijacking/.

[8] MDN Web Docs - X-Forwarded-For. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Forwarded-For.

[9] MiniOrange. https://www.miniorange.com/.

[10] OpenWPM. https://github.com/mozilla/OpenWPM.

[11] OWASP -Top 10 Web Application Security Risks. https://owasp.org/www-project-top-ten/.

[12] THE APWG ECRIME EXCHANGE (ECX). https://apwg.org/ecx/.

[13] CyDec Security Anti-Fp, 2021. https://chrome.google.com/webstore/detail/cydec-security-anti-fp/becfjfjckdhngmmpkhakoknnkgpgfelk.

[14] DUO - The 2021 State of the Auth Report: 2FA Climbs, While Password Managers and Biometrics Trend, 2021. https://duo.com/blog/the-2021-state-of-the-auth-report-2fa-climbs-password-managers-biometrics-trend.

[15] User-Agent Switcher and Manager, 2021. https://chrome.google.com/webstore/detail/user-agent-switcher-and-m/bhchdcejhohfmigjafbampogmaanbfkg.

[16] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14.

[17] Furkan Alaca and P. C. van Oorschot. Device fingerprinting for augmenting web authentication: Classification and analysis of methods. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16.

[18] Nampoina Andriamilanto and Tristan Allard. Brfast: A tool to select browser fingerprinting attributes for web authentication according to a usability-security trade-off. In *Companion Proceedings of the Web Conference 2021*, WWW '21, page 701–704, 2021.

[19] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. "guess who?" large-scale data-centric study of the adequacy of browser fingerprints for web authentication. In *Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 161–172, 2021.

[20] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. FPSelect: Low-Cost Browser Fingerprints for Mitigating Dictionary Attacks against Web Authentication Mechanisms. In *Annual Computer Security Applications Conference (ACSAC 2020)*, 2020.

[21] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. Disguised chromium browser: Robust browser, flash and canvas fingerprinting protection. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, 2016.

[22] Joseph Bonneau, Edward W Felten, Prateek Mittal, and Arvind Narayanan. Privacy concerns of implicit secondary factors for web authentication. *WAY*, 14, 2014.

[23] Elie Bursztein, Borbala Benko, Daniel Margolis, Tadek Pietraszek, Andy Archer, Allan Aquino, Andreas Pitsillidis, and Stefan Savage. Handcrafted fraud and extortion: Manual account hijacking in the wild. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14.

[24] Michele Campobasso and Luca Allodi. Impersonation-as-a-service: Characterizing the emerging criminal infrastructure for user impersonation at scale. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[25] Yinzhi Cao, Song Li, and Erik Wijmans. (cross-)browser fingerprinting via os and hardware level features. In *Proceedings of Network & Distributed System Security Symposium (NDSS)*. Internet Society, 2017.

[26] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS*, volume 14, pages 23–26, 2014.

[27] Emiliano De Cristofaro, Honglu Du, Julien Freudiger, and Greg Norcie. A comparative usability study of two-factor authentication. *arXiv:1309.5344*, 2013.

[28] Joe DeBlasio, Stefan Savage, Geoffrey M. Voelker, and Alex C. Snoeren. Tripwire: Inferring internet site compromise. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17.

[29] Alexandra Dmitrienko, Christopher Liebchen, Christian Rossow, and Ahmad-Reza Sadeghi. On the (in) security of mobile two-factor authentication. In *International Conference on Financial Cryptography and Data Security*. Springer, 2014.

[30] Periwinkle Doerfler, Maija Marincenko, Juri Ranieri, Yu Jiang, Angelika Moscicki, Damon McCoy, and Kurt Thomas. Evaluating login challenges as a defense against account takeover. In *Proceedings of the International Conference on World Wide Web*, 2019.

[31] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. The cookie hunter: Automated black-box auditing for web authentication and authorization flaws. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20.

[32] Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, 2010.

[33] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, 2016.

[34] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W. Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15.

[35] D. Freeman, Sakshi Jain, Markus Dürmuth, B. Biggio, and G. Giacinto. Who are you? a statistical approach to measuring user authenticity. In *Proceedings of Network & Distributed System Security Symposium (NDSS)*. Internet Society, 2016.

[36] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis. O single {Sign-Off}, where art thou? an empirical analysis of single {Sign-On} account hijacking and session management on the web. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[37] David Goltzsche, Colin Wulf, Divya Muthukumaran, Konrad Rieck, Peter Pietzuch, and Rüdiger Kapitza. Trustjs: Trusted client-side execution of javascript. In *EuroSec 2017*, pages 1–6, 2017.

[38] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *WWW 2018*.

[39] Adam Hurkała and Jarosław Hurkała. Architecture of context-risk-aware authentication system for web environments. In *Proceedings of the Third International Conference on Informatics Engineering and Information Science (ICIEIS)*, 2014.

[40] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. *arXiv preprint arXiv:2008.04480*, 2020.

[41] Roger Piqueras Jover. Security analysis of sms as a second factor of authentication. *Communications of the ACM*, 63(12), 2020.

[42] Jordan Jueckstock and Alexandros Kapravelos. VisibleV8: In-browser Monitoring of JavaScript in the Wild. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2019.

[43] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. Carnus: Exploring the privacy threats of browser extension fingerprinting. In *27th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2020.

[44] Pierre Laperdrix, Gildas Avoine, Benoit Baudry, and Nick Nikiforakis. Morellian Analysis for Browsers: Making Web Authentication Stronger with Canvas Fingerprinting. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 16th International Conference, DIMVA*. 2019.

[45] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)*, 14(2).

[46] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016.

[47] Sebastian Lekies, Ben Stock, and Martin Johns. 25 million flows later: large-scale detection of dom-based xss. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1193–1204, 2013.

[48] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security 16)*.

[49] Xu Lin, Panagiotis Ilia, and Jason Polakis. Fill in the blanks: Empirical analysis of the privacy threats of browser form autofill. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[50] Zengrui Liu, Prakash Shrestha, and Nitesh Saxena. Gummy Browsers: Targeted Browser Spoofing against State-of-the-Art Fingerprinting Techniques. In *In International Conference on Applied Cryptography and Network Security (ACNS)*, 2022.

[51] Ariana Mirian, Joe DeBlasio, Stefan Savage, Geoffrey M. Voelker, and Kurt Thomas. Hack for hire: Exploring the emerging market for account hijacking. In *The World Wide Web Conference*, WWW '19.

[52] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in html5. *Proceedings of W2SP*, 2012.

[53] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013.

[54] Sean Oesch and Scott Ruoti. That was then, this is now: A security evaluation of password generation, storage, and autofill in Browser-Based password managers. In *29th USENIX Security Symposium*, 2020.

[55] Jeremiah Onaolapo, Enrico Mariconti, and Gianluca Stringhini. What happens after you are pwnd: Understanding the use of leaked webmail credentials in the wild. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16.

[56] OWASP. Credential stuffing prevention cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html, 2021.

[57] Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Alain Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17.

[58] Iasonas Polakis, Marco Lancini, Georgios Kontaxis, Federico Maggi, Sotiris Ioannidis, Angelos D Keromytis, and Stefano Zanero. All your face are belong to us: Breaking facebook's social authentication. In *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012.

[59] Ken Reese, Trevor Smith, Jonathan Dutson, Jonathan Armknecht, Jacob Cameron, and Kent Seamons. A usability study of five Two-Factor authentication methods. In *SOUPS 2019*.

[60] Richard Shay, Iulia Ion, Robert W. Reeder, and Sunny Consolvo. "my religious aunt asked why i was trying to sell her viagra": Experiences with account hijacking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14.

[61] David Silver, Suman Jana, Dan Boneh, Eric Chen, and Collin Jackson. Password managers: Attacks and defenses. In *23rd USENIX Security Symposium*, 2014.

[62] Suphannee Sivakorn, Angelos D Keromytis, and Jason Polakis. That's the way the cookie crumbles: Evaluating https enforcing mechanisms. In *ACM WPES*, 2016.

[63] Suphannee Sivakorn, Jason Polakis, and Angelos D. Keromytis. The cracked cookie jar: Http cookie hijacking and the exposure of private information. In *In Proceedings of the 37th IEEE Symposium on Security and Privacy*, S&P '16.

[64] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. Discovering browser extensions via web accessible resources. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17.

[65] Jan Spooren, Davy Preuveneers, and Wouter Joosen. Mobile device fingerprinting considered harmful for risk-based authentication. In *Proceedings of the Eighth European Workshop on System Security*, EuroSec '15, 2015.

[66] Oleksii Starov and Nick Nikiforakis. Xhound: Quantifying the fingerprintability of browser extensions. In *2017 IEEE Symposium on Security and Privacy (SP)*.

[67] Roland H. Steinegger, Daniel Deckers, Pascal Giessler, and Sebastian Abeck. Risk-based authenticator for web applications. In *Proceedings of the 21st European Conference on Pattern Languages of Programs*, EuroPlop '16.

[68] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, Daniel Margolis, Vern Paxson, and Elie Bursztein. Data breaches, phishing, or malware? understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17.

[69] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan

Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*.

[70] Steven Van Acker, Daniel Hausknecht, and Andrei Sabelfeld. Measuring login webpage security. In *SAC 2017*.

[71] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. Fp-stalker: Tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018.

[72] Stephan Wiefling, Markus Dürmuth, and Luigi Lo Iacono. What's in Score for Website Users: A Data-driven Long-term Study on Risk-based Authentication Characteristics. In *25th International Conference on Financial Cryptography and Data Security (FC '21)*.

[73] Stephan Wiefling, Markus Dürmuth, and Luigi Lo Iacono. More than just good passwords? a study on usability and security perceptions of risk-based authentication. In *ACSAC*, 2020.

[74] Stephan Wiefling, Luigi Lo Iacono, and Markus Durmuth. Is this really you? an empirical study on risk-based authentication applied in the wild. *IFIP AICT*, 2019.

[75] Stephan Wiefling, Tanvi Patil, Markus Dürmuth, and Luigi Lo Iacono. Evaluation of Risk-Based Re-Authentication Methods. In *ICT Systems Security and Privacy Protection*, 2020.

[76] Penghui Zhang, Adam Oest, Haehyun Cho, Zhibo Sun, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kapravelos, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. CrawlPhish: Large-scale Analysis of Client-side Cloaking Techniques in Phishing. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2021.

A Use Cases

We provide additional details about select cases of websites susceptible to our attack. Our comprehensive manual analysis helped us understand how risk-based authentication systems behave and revealed shortcomings in their implementation.

`Tax-B` does not enable 2FA by default. Instead, the user can opt in through the website's settings by selecting email, SMS, or the Google Authenticator app as their second factor. `Tax-B` also records the user's IP address for each login. Users can find information about their trusted devices and past logins in the account preferences, including sign-in timestamps as well as the devices' IP addresses. Interestingly, while `Tax-B` records the IP addresses used in past logins, we found that it does not use this information to determine if a new login is suspicious. Our experiments reveal that `Tax-B` uses a fingerprinting script on its login page that is highly similar to the popular fingerprinting library `FingerprintJS`. A notable difference, however, is that `Tax-B`'s script does not implement audio fingerprinting. Furthermore, by inspecting outgoing network traffic when logging into the website using our primary device, we observed that a JSON string that includes 33 fingerprinting values is sent to `Tax-B`'s server. To fingerprint JavaScript fonts, `Tax-B` uses "monospace" as the base font and checks against a list of 495 different font families. After embedding `Tax-B`'s code and fonts list into our phishing site and visiting it with our primary device, we found that our device supports 88 font families. Then, to spoof the font fingerprint during the attack phase, our extension changes the `offsetWidth` and `offsetHeight` properties of the span elements that load these 88 fonts to deviate from their default values.

`eCommerce-A` also does not use 2FA by default and users need to explicitly enable it. Their site has multiple fingerprinting scripts on the login page that implement both basic and advanced fingerprinting techniques, such as canvas, WebGL, fonts and audio fingerprinting. Our analysis revealed that `eCommerce-A` does not actually use fonts and audio fingerprints for authentication, but only relies on basic fingerprints, canvas, and WebGL. Regarding their basic fingerprints, we observed that `eCommerce-A` collects 46 attributes, such as `Navigator.plugins` and `Window.devicePixelRatio`. For canvas and WebGL fingerprinting, `eCommerce-A` uses two different scripts and draws 7 images in total. In two of the images it draws a string of a random integer number between 0 and 999. As described previously, our phishing site records the Base64 values of the images and then, during the attack, our extension manipulates the `toDataURL()` method to return the Base64 strings (in the correct order). For fingerprinting fonts, `eCommerce-A` uses "'monospace", "sans-serif" and "serif" as base fonts and compares against 485 font families. In comparison to `Tax-B` that uses one base font and loads each font family to be tested in a single span element, `eCommerce-A` uses three different base fonts as a fallback, loading each font family to be tested in three different span elements, and checking for changes in the dimensions of any of these elements.

`RideSharing` exhibits a unique idiosyncrasy as it exposes two login URLs, which actually behave differently. Specifically, any login attempts made from the landing page that do not include the necessary cookies will always result in 2FA being triggered. Surprisingly, the login attempts from the other page will only trigger 2FA when a new device is used. As such, attackers can impersonate the user's device and log in from this second page to bypass 2FA. We also found that if we explicitly enable 2FA through `RideSharing`'s mobile app, 2FA is triggered for every login attempt regardless of which login URL is used. `RideSharing` collects a total of 379 fingerprinting attributes from the user's device. More interestingly, it employs a fingerprinting strategy that we have not come across in other sites. Specifically, it catches errors

during each step of the fingerprinting process and pushes those errors into an array which is used to calculate one of the fingerprinting values. For example, the script tries to create an element for obtaining a list of fonts that are available in Internet Explorer but not supported by other browsers.

B Inconsistency Checks

Tax-A uses `toString()` to detect if any native functions have been tampered with. When `toString()` is called on a function, it returns a string representation of the function's code. In the case of native functions, the returned value shows that the function uses native code. When a function is overridden (by our extension) to return a spoofed value, its string representation returned by `toString()` would reveal this change. For bypassing such checks, we also override the `toString()` method to make it return the *expected* value for native methods.

Tax-B checks whether the browser's languages have been tampered with by comparing the `Navigator.language` attribute with `Navigator.languages`. It also compares the screen's size with the available size. Finally, it determines the browser type based on the user-agent and looks for contradictions between the browser type and `Navigator.productSub` or `eval.toString().length`. Similarly to `Bank-A`'s case, our attack is not affected by these inconsistency checks as we spoof these attributes according to the primary device's values.

RideSharing catches JavaScript runtime exceptions and uses the error messages as fingerprints. If the attacker's browser is different from the victim's, these error messages will differ. However, the attacker can hook the specific APIs in relation to these errors, and change them to show custom error messages. For example, `RideSharing`'s code creates an element that throws an exception in modern browsers By hooking `document.createElement()`, the attacker can make it throw a custom error message that looks like the one shown by the victim's browser when such an element is created.

C Properties of JavaScript Objects

In Table 6 we present the fingerprintable properties of the `Navigator`, `Window`, `Screen`, `Plugin`, and `MimeType` objects that our extensions can obtain and spoof. We decided on this set of properties as these are used by `Fingeprintjs2` [5], the extended version of `OpenWPM` presented in [40], and also in fingerprinting scripts we found during our exploration of high-value services. Properties marked with '*' are only supported in Internet Explorer, but they are still widely used in phishing websites.

D Canvas Fonts Fingerprinting

The `CanvasRenderingContext2D.measureText()` method returns a `TextMetrics` object that contains information about the measured text (such as its width) that is rendered on the canvas. If a tested font is not supported, the default fallback font is used instead. The fingerprinting script loops through a list of fonts and measures their rendered width. If the baseline width and the tested font's width are equal, it means that the particular font is not supported by the browser. On our primary device, we hook the `measureText()` method and collected all the `TextMetrics` objects. Then, on our secondary device, our extension modifies the `measureText()` method to replace the returned values with those collected from the primary device.

Table 6: Fingerprintable properties of JavaScript objects. Properties marked with '*' are only supported in Internet Explorer.

JavaScript Object	Fingerprintable Properties
`Navigator`	userAgent, platform vendor, vendorSub, product, productSub, oscpu, cpuClass*, buildID, hardwareConcurrency, appName, appCodeName, appVersion, appMinorVersion*, languages, language, browserLanguage*, userLanguage*, systemLanguage*, permissions, onLine, connection, cookieEnabled, doNotTrack, deviceMemory, getBattery, geolocation, getGamepads, maxTouchPoints, msMaxTouchPoints* mediaDevices, mimeTypes, javaEnabled, plugins, sendBeacon, vibrate, bluetooth, webdriver
`Window`	innerWidth, innerHeight, outerWidth, outerHeight, screenLeftm screenTop, screenX, screenY, devicePixelRatio, ontouchstart, swfobject, ActiveXObject*, locationbar, menubar, toolbar, statusbar, personalbar, scrollbar, pageXOffset, scrollX, speechSynthesis, sessionStorage, localStorage, indexedDB, openDatabase
`Screen`	width, height availWidth, availHeight, availLeft, availTop, colorDepth, pixelDepth, deviceXDPI*, systemXDPI*, logicalXDPI*, deviceYDPI*, logicalYDPI*, updateInterval*, orientation
`Plugin`	name, version, description, filename
`mimeType`	type, description, enabledPlugin, suffixes

E Phishing Sites: Fingerprint Exfiltration

We manually examined our `VisibleV8` logs for 500, 500, and 200 sites targeting `Bank-A`, `RideSharing`, and `WebInfrastructure` respectively. We found that phishing sites use APIs like `XMLHttpRequest` and `WebSocket` for sending fingerprinting values back to the server. Some sites also include fingerprinting values in the URLs of `GET` requests. We found 197, 109, and 4 phishing sites respectively exfiltrating fingerprinting values. Additionally, 164, 128, and 126 sites send back obfuscated data. While this likely includes fingerprinting values in certain cases, the costly manual process required to verify this falls outside the scope of our work.