Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods

Furkan Alaca P.C. van Oorschot

School of Computer Science
Carleton University, Ottawa, Canada

ABSTRACT

Device fingerprinting is commonly used for tracking users. We explore device fingerprinting but in the specific context of use for augmenting authentication, providing a state-of-the-art view and analysis. We summarize and classify 29 available methods and their properties; define attack models relevant to augmenting passwords for user authentication; and qualitatively compare them based on stability, repeatability, resource use, client passiveness, difficulty of spoofing, and distinguishability offered.

CCS Concepts

•Security and privacy → Authentication; Web application security;

Keywords

Device fingerprinting, multi-dimensional authentication, passwords, comparative analysis, comparative criteria

1. INTRODUCTION

We explore the state-of-the-art of device fingerprinting, albeit with a special focus: to augment web authentication mechanisms, and especially password-based methods. Despite usability and security drawbacks, the latter remains the dominant form of authentication on the web—in part because more secure alternatives have their own usability and deployability issues [13].

Device fingerprinting involves techniques by which a server collects information about a device's software and/or hardware configuration for the purpose of identification. Web browsers reveal information to websites about the host system both explicitly by exposing data such as screen resolution, local time, or OS version, and implicitly by leaking information about the device's software or hardware configuration through observable differences in browser behaviour. Web analytics and advertising services are known to employ browser-based device fingerprinting to track users' browsing habits [21, 2]. In addition, a number of commercial fraud detection services [38, 57] employ browser-based device fingerprinting to identify fraudulent transactions. Large web services can mitigate insecure passwords in part by using server-side intelligence in the authentication process with "multidimensional" authentication [14], wherein conventional passwords are augmented by multiple implicit signals collected from the user's device without requiring user intervention. We explore the applicability of various fingerprinting techniques to strengthening web authentication and evaluate them based on the security benefits they provide. We aim to identify techniques that can improve security, while being ideally invisible to users and compatible with existing web browsers, thereby imposing zero cost on usability and deployability.

We identify and classify 29 available device fingerprinting mechanisms, primarily browser-based and known, but including several network-based methods and others not in the literature; identify and assess their properties suitable for application to augment user authentication; define a series of adversarial models within the context of fingerprint-augmented authentication; and consider practical issues in the implementation, deployment, and use of device fingerprinting within the context of user authentication.

While prior work has mostly presented specific device fingerprinting mechanisms or their use by web trackers, we analyze a broad range of such mechanisms within the context of augmenting password-based web authentication.

ACSAC '16, December 05-09, 2016, Los Angeles, CA, USA

 ISBN 978-1-4503-4771-6/16/12…$15.00

DOI: http://dx.doi.org/10.1145/2991079.2991091

2. BACKGROUND AND RELATED WORK

Eckersley [21] published the first research paper discussing in detail the concept of browser-based device fingerprinting and showed that websites could identify users and track their browsing habits by collecting basic information such as device IP address, time zone, screen resolution, and a list of supported fonts and plugins; such functionality was originally designed to help optimize web content for a wide variety of target devices. Mowery et al. [41, 42] later proposed two more advanced fingerprinting methods; the first measures the performance signature of the browser's JavaScript engine, and the second renders text in an HTML5 canvas to distinguish font-rendering techniques across different software and hardware platforms. Nikiforakis et al. [48] discuss the difficulties of thwarting fingerprinting techniques with client-side privacy extensions; e.g., differences in JavaScript implementations across browser vendors can reveal which browser is being used even if the user-agent string is obscured; web proxies can often be circumvented to reveal

the user's true IP address through external plugins such as Flash or Java (cf. [44]); and user manipulation of the device fingerprint may result in a less common fingerprint that is even more useful for identifying the user. Bojinov et al. [11] demonstrate how smartphones can be fingerprinted via a multitude of onboard sensors; in particular, they use the device's accelerometer calibration error and the frequency response of the speakerphone-microphone system. Nikiforakis et al. [47] developed PriVaricator, which reduces fingerprintability by randomizing various browser parameters.

Empirical studies by Acar et al. [3, 2] and Nikiforakis et al. [48] reveal extensive use of device fingerprinting by advertisers as a fallback mechanism to track users, should they clear their browser cookies. Advertisers also save identifying information via less conventional storage mechanisms, e.g., Flash cookies, which are more difficult for users to delete [39]; this information may be used to reconstruct browser cookies, should the user clear them [54, 6, 40, 2].

Some work has begun to explore using device fingerprinting to enhance web security. Unger et al. [58] propose enhancing session security through server-side monitoring of certain web browser attributes (e.g., user-agent string, supported CSS features) to help detect session hijacking. Preuveneers and Joosen [51] propose a protocol that monitors various parameters throughout an authenticated session, such as user IP address and time of access; applies a similarity-preserving hash function to enable privacy-preserving fingerprint storage that allows similarity checks between stored fingerprints and subsequently-collected fingerprints; and uses a comparison algorithm that assigns a weight to each attribute (based on its usefulness for identifying that particular client) to determine if the fingerprint has changed significantly enough to ask the user to re-authenticate. Van Goethem et al. [59] propose an accelerometer-based device fingerprinting mechanism for multi-factor mobile authentication. Freeman et al. [27] propose a statistical framework to detect suspicious login attempts using various browser attributes and other parameters such as time of access.

Spooren et al. [55] argue that the relative lack of diversity in mobile device fingerprints, compared to desktop computers, makes them less reliable for risk-based authentication. However, the study did not include many of the more recent and advanced fingerprinting mechanisms discussed herein. Laperdrix et al. [36] collected and analyzed 119,000 fingerprints, and found that while some mobile device attributes are less diverse (e.g., limited browser plugin support), other attributes (e.g., user-agent string and canvas fingerprinting) are much more diverse than on desktops. Eisen [22, 23] is the inventor on two patents relevant to device fingerprinting: the first describes how a server may detect fraudulent transactions by recording the difference between the server's local time and that of each client (to determine the client's time zone, observance of daylight saving time, and drift from UTC); the second describes a generalized framework wherein a server can obtain a fingerprint on each page that the client requests from the server and signal a warning for a session-tampering attempt when there is sufficient change in the fingerprint. A patent by Varghese et al. [60] describes how a client's device fingerprint can be used as an index to retrieve its associated risk of fraud from a central database of fingerprints for devices suspected of participating in fraud, and accordingly grant the client an appropriate level of access to the account.

Implicit authentication [31] is a related technique that aims to strengthen or replace existing authentication mechanisms by relying on user behaviour such as typing rhythms, touchscreen input, sensor input (e.g., gyroscope, accelerometer, nearby Bluetooth or WiFi devices), usage patterns (e.g., time of access, commonly accessed pages), etc. While some of these techniques are also applicable to the web, our focus herein is on leveraging device information rather than user behaviour. However, in the broader context of "multidimensional" authentication, user behaviour profiling can be used alongside device fingerprinting (and potentially other implicit signals) to achieve stronger authentication guarantees without sacrificing usability.

3. FRAMEWORK AND THREAT MODEL

Base definitions, replay and spoofing. In device fingerprinting, servers use one or more mechanisms (*fingerprinting vectors* or *vectors*) to extract and verify properties related to the software and/or hardware configuration of a device. "Pure" device fingerprinting is *stateless*, depositing no new client-side information; alternatives are *stateful*. By *device fingerprint* we mean the overall set of vectors a server employs, or depending on context, the output corresponding to such vectors. A base assumption is that attackers eventually learn the set of vectors composing a device fingerprint. A server may use a collection of vectors and choose some (random or other) subset in particular instances; if so, we assume the attacker does not know the subset beforehand.

Many vectors require browsers to perform certain operations, e.g., via JavaScript, and return the output to the server. If a vector response is *static*, i.e., constant regardless of circumstances, an attacker observing or intercepting this can later *replay* it. If all vectors composing a device fingerprint are static, so is the device fingerprint, and simple replay suffices. By *spoofing* we mean an attacker trying to mimic the fingerprint of a target device; this is a form of *mimicry attack*. If a server varies the vectors, or uses individual vectors for which the response is dependent on conditions or a variable challenge of some form, then spoofing requires a more complex attack than simple capture and replay—e.g., successfully spoofing some forms of geolocation (as discussed later) requires attacker access to a proxy machine located near the victim. In the models below and elsewhere, we typically use the more general term spoofing, but often this requires only replay.

To facilitate analysis of device-fingerprinting-augmented authentication, we define a continuum of five attack models, in order of increasing attacker power and skill. Table 1 summarizes and compares attacker capabilities across these models. This modelling also serves to motivate and focus later discussion of desirable defensive properties of fingerprinting vectors (see Section 5 and Table 2).

Model M1: Naive attack. We consider a naive attack to be a conventional online password-guessing attack that does not attempt to thwart any secondary authentication mechanisms. This is the easiest attack to protect against: if the attacker attempts to guess an account password while making no effort to spoof the device fingerprint of the account owner, the server can determine based on historical data that the authentication attempt is from a device that the account owner has never used. Advanced attackers will use the more sophisticated models below.

Attack models	Optimized password-guessing	Optimized device fingerprint guessing	Obtain partial or full target device details	Phish victim password and device fingerprint	Spoof device fingerprint	Steal session cookie
M1: Naive attack	●					
M2: Optimized password and fingerprint guessing	●	●			●	
M3: Targeted password and fingerprint guessing	●	●	●		●	
M4: Fingerprint phishing and spoofing			●	●	●	
M5: Session hijack with fingerprint spoofing			●		●	●

Table 1: Attacker models (rows) and capabilities (columns).

Model M2: Optimized password and fingerprint guessing attack. This attack follows an optimal guessing strategy, i.e., guessing passwords in order of popularity across many accounts [12] while iterating over device fingerprint "guesses" expected to maximize the probability of account break-in. The attacker may spoof or mimic several fingerprints that match a set of popular devices, potentially tailored to the target website audience; e.g., websites targeting users in particular countries are likely to have visitors within certain time zones, or vendor-specific technical support sites will likely have visitors using that vendor's devices. The attacker can then, based on the popularity of each password and device fingerprint combination, mount an attack by iterating through the list of password-fingerprint pairs in order of decreasing popularity across all accounts. In special cases, the attacker may reduce the search space by obtaining a list of passwords (e.g., from a leaked password database of another website, if users can be mapped between the two websites through information such as e-mail addresses) or a list of device fingerprints (e.g., by exploiting a XSS vulnerability to obtain the necesary data from a large subset of users to spoof their device fingerprints).

Model M3: Targeted password and fingerprint guessing attack. The attacker aims to break into a specific account, and mounts a password guessing attack while also attempting to spoof a fingerprint to match a target device. This attack iterates over passwords and fingerprints in some optimized order, as in M2. The attacker may significantly reduce the search space by using known information (e.g., if the user is known to use a particular type of device) or in special cases may possess (e.g., due to *a priori* capture) the user's password or a fingerprint of the user's device, thereby reducing the attack to either fingerprint guessing or password guessing.

Model M4: Fingerprint phishing and spoofing. Whereas classic phishing attacks steal passwords, a related attack could target websites using device-fingerprinting-augmented authentication to capture both the victim's password and device fingerprint. The attacker will then spoof the fingerprint and access the account with the captured password. This attack is harder if the device fingerprint is not easily spoofed.

Model M5: Session hijack with fingerprint spoofing. The attacker aims to hijack an existing authenticated session, vs. attackers M1-M4 which attempt account break-in via password and fingerprint guessing or stealing. The M5 attacker is granted the power to steal session cookies and execute client-side JavaScript by exploiting, e.g., improperly-configured HTTPS [53, 35] or XSS vulnerabilities. This enables the attacker to (1) capture device fingerprints sent from the browser to the server, and (2) perform any additional fingerprinting that would facilitate spoofing the user's device and thereby resuming the session from the attacker's device. A device fingerprint useful under M5 must thus be hard to spoof (beyond simply replaying a static string). Server-side fingerprint checking has been proposed as a defence against attacker reuse of stolen session cookies [58, 51].[1]

We now provide some simple comments on these models:

- Attacks under M1: Device fingerprinting completely stops naive attacks, even if fingerprints are replayable.
- Attacks under M2: Device fingerprinting significantly reduces the success probability of M2 attacks, even with replayable fingerprints, if the guessing space of the fingerprints is sufficiently large relative to the number of guesses the attacker is capable or allowed to submit.
- Attacks under M3: These attacks are more difficult to defend against, since the attacker targets a specific user and is assumed to have device-specific information.
- Attacks under M4: These are the most difficult to prevent among M1-M4, since the M4 attacker lures the user by phishing to capture a password and device fingerprint. If the fingerprint is replayable, this suffices for spoofing.
- M5 is specific to session hijacking, with fingerprinting used not simply to augment authentication at the start of a session, but throughout one; this is pursued in Section 6.2.

The above models and discussion help delineate properties important in our later analysis and classification.

4. DEVICE FINGERPRINTING VECTORS: CLASSIFICATION

We now identify, summarize, and classify fingerprinting vectors based on a review of research literature, informal sources (e.g., online sources, open-source fingerprinting libraries, online advertising and anti-fraud services), and other fairly obvious vectors including some not formally documented elsewhere to our knowledge. In most cases, we classify these vectors based on the method used to obtain the fingerprintable information from the client. Alternative classifications are possible, such as based on the type of information collected: hardware features, network information, user-defined preferences, and software installed (e.g., OS, applications, device drivers, shared libraries).

[1]The motivation behind this use case is similar to that of Channel ID (originally known as Origin-Bound Certificates [20]), which uses self-signed, domain-specific SSL client certificates dynamically generated by the client. Session cookies are cryptographically bound to the SSL session, thereby requiring an attacker to possess both the client's cookie and domain-specific private key to hijack a session.

Category 1: Browser-provided information. Web browsers explicitly provide (e.g., via JavaScript) a wide range of system information that is of use to constructing a device fingerprint. Known vectors are listed below.

(a) **Major software and hardware details.** The navigator and window Browser Object Model (BOM) objects expose attributes such as browser/OS vendor and version, system language, platform, user-agent string (which includes the prior three and sometimes others, e.g., device model number), installed plugins, supported browser storage mechanisms (e.g., localStorage, indexedDB, sessionStorage, WebSQL via openDatabase), screen resolution, colour depth, and pixel density.

(b) **WebGL information.** WebGL [43], a JavaScript API for rendering graphics within web browsers, exposes various attributes (e.g., GL version, max texture size or render buffer size, supported WebGL extensions, vendor/renderer strings) of the underlying browser and hardware.

(c) **System time and clock drift.** The device's system time can be accessed via JavaScript and used to infer the device's time zone, observance of daylight saving time, and clock drift from Coordinated Universal Time (UTC) [22].

(d) **Battery information.** The HTML5 battery status API is suitable for fingerprinting when sufficiently precise readouts are provided [50]. The battery's charge level can be used for short-term tracking of clients across different websites; its capacity (which slowly degrades with battery age, but changes very little over relatively short periods such as days) can be estimated by monitoring the discharge rate for about 30 seconds, and used to aid identification.

(e) **Evercookies.**[2] An *evercookie* is a mechanism whereby a client identifier (and/or other tracking information commonly stored in a browser cookie) is stored on the device using a variety of techniques such as HTML5 local storage, HTTP ETags, or Flash cookies, thereby allowing websites to reconstruct user-deleted cookies [2].

(f) **WebRTC.** WebRTC is a set of W3C standards that support native (plugin-free) browser-to-browser applications such as voice and video chat [8]. Devices can be fingerprinted by enumerating supported WebRTC features and potentially connected media devices such as microphones and webcams; browsers differ as to what types of devices can be enumerated without user permission. WebRTC also exposes the IP addresses assigned to all network interfaces present on the device [52], including private IP addresses assigned by a NAT router or VPN.

(g) **Password autofill.** JavaScript can be used to detect if a password has been user-typed, or auto-filled by a browser or password manager. Our tests show that this is easily achieved using event listeners to detect whether the user has typed any characters into the password field, e.g., by assigning an event listener for the keydown event (triggered when a key is pressed down) and keypress event (triggered after a key has been pressed and subsequently released). Since these events are triggered by physical key presses, their absense indicates password entry via autofill (or other automated means, e.g., an automated password-guessing attack).

Category 2: Inference based on device behaviour. Information about the device can be gleaned not only by querying the browser (as in Category 1), but also by executing specially-crafted JavaScript code on the browser and observing the effect, e.g., measuring execution time or analyzing generated output. The following fall in this category.

(a) **HTML5 canvas fingerprinting.** JavaScript can be used client-side to render a variety of text and graphics in an HTML5 canvas, and send the server a hash of the resulting bitmap image [42]. Subtly different images are generated by devices with different software/hardware configurations, e.g., fonts and font rendering (e.g., anti-aliasing) vary with OS and video driver, and emojis vary with OS and smartphone vendor [36]. Rendering text using a list of predefined fonts allows font detection [24]. Rendering complex graphics using WebGL provides further fingerprint diversity by a wide range of rendered hardware-dependent graphics output.

(b) **System performance.** Client-side JavaScript can be used to run JavaScript engine benchmarks over a range of computationally-intensive operations. The elapsed times are measured to infer device performance characteristics [41].

(c) **Hardware sensors.** Mobile device sensors can be fingerprinted based on variations in manufacturing and factory calibration, e.g., measuring calibration error of smartphone accelerometers (accessible via JavaScript) or the frequency response of speakerphone-microphone systems [11, 19].

(d) **Scroll wheel fingerprinting.** Various aspects of user pointing devices can be inferred via JavaScript by listening for the WheelEvent event, triggered whenever the user scrolls using a mouse wheel or touchpad [49]. For example, a mouse wheel scrolls the page by fixed increments when triggered; a touchpad can scroll by varying increments. Measuring the rate at which a document is scrolled reveals information both about user scrolling behaviour and the value of the OS user-configurable scrolling speed.

(e) **CSS feature detection.** The browser vendor and version can be inferred by testing CSS features that are not uniformly supported across browsers [58], e.g., by setting the desired CSS property on a target element and subsequently querying the element to determine if the change was applied. This vector (and vectors 2f and 2g, below) yields a subset of the information obtained from the user-agent string (via 1a or 4e). If a device fingerprint already extracts user-agent string information by another vector, then 2e here can also be used to test if that information has been manipulated.

(f) **JavaScript standards conformance.** Browsers differ in conformance to the JavaScript standard, allowing fingerprinting based on behaviour in corner cases. Various JavaScript conformance tests run thousands of test cases that can collectively take over 30 minutes. Mulazzani et al. [45] developed a technique that leverages the fact that modern browsers fail very few of these tests. This technique can be used to verify the browser vendor and version reported in the user-agent string by using a decision tree to select a very small subset of tests that run in negligible time.

(g) **URL scheme handlers.** Some browsers implement non-standard schemes to access local resources; e.g., res:// in Microsoft IE exposes images stored in DLL files in the Windows system directory, and moz-icon://, jar:resource://

[2]Evercookies violate the stateless property of our "pure" device fingerprinting definition, but are nonetheless considered a form of fingerprinting by, e.g., the W3C: https://w3c.github.io/fingerprinting-guidance

and resource:// in Mozilla Firefox expose various built-in browser and OS resources. Although recent browser versions disallow websites from accessing local files via file:// (to prevent leakage of user private data), this restriction does not extend to the aforementioned schemes. Websites can thus create HTML image tags with the source address set to a local resource and use the onerror event handler to detect if the image was loaded. By iterating through a list of resources preloaded with various browser or OS versions, this vector can enumerate the non-standard schemes supported and the resources successfully loaded [30]. This offers an alternative (to 2e, 2f, and 1a) to infer browser vendor and version.

(h) **Video RAM detection.** The amount of Video RAM (VRAM) available to the GPU, while not explicitly available via the WebGL API, can be inferred by repeatedly allocating textures until VRAM is full, after which textures start being swapped into system main memory. By observing the elapsed time for each texture allocation and recording the step at which a large spike in elapsed time is observed,[3] it can be inferred that GPU VRAM is at full utilization. After this point, the browser can continue allocating textures until an OUT_OF_MEMORY[4] error is returned.

(i) **Font detection.** While installed fonts cannot be enumerated via JavaScript, text can be formatted with fonts from a predefined list; the dimensions of the resulting text can distinguish different font rendering settings and allow the presence of each font (and even versions of the same font) to be inferred [48, 25].

(j) **Audio processing.** The HTML5 AudioContext API allows the creation of audio visualizations by providing an interface for real-time frequency- and time-domain analysis of audio playback. As with 2(a) HTML5 canvas fingerprinting, audio processing varies by browser and software/hardware configuration [24].

Category 3: Browser extensions and plugins. Browser plugins and extensions can be leveraged for fingerprinting in a manner similar to Categories 1, 2, and 4 (i.e., by directly querying for information, by inference, and/or by protocol-level analysis). The following are such methods.

(a) **Browser plugin fingerprinting.** Browser plugins such as Java, Flash, and Silverlight can be queried (via plugin objects embedded in a webpage) to reveal system information, often in more detail than available via JavaScript. For example, Flash provides the full OS kernel version; both Flash and Java plugins allow all system fonts to be enumerated; even the order in which system fonts are enumerated can vary across systems, increasing the distinguishability of fingerprints [48, 21].

(b) **Browser extension fingerprinting.** If the NoScript extension is installed (disabling JavaScript by default for all websites, except those whitelisted by the user), a website can attempt to load scripts from a large set of websites (e.g., Alexa Top 1000) to detect which are on the user whitelist [41]. Similarly, ad blockers can be detected by embedding a "fake" ad such as a hidden image or iframe with a source URL containing words commonly blacklisted by ad blockers (e.g., "ads"); JavaScript can then detect if the fake ad has been loaded, and return the result to the server. Other extensions may be fingerprinted by other methods, e.g., some browser extensions add custom HTTP headers [36].

(c) **System-fingerprinting plugins.** Websites may install specialized plugins (e.g., by bundling them with other software available for download) to provide more powerful fingerprinting information such as hardware identifiers, OS installation date, and installed driver versions [48]. Such plugins are generally considered *spyware*.

Category 4: Network- and protocol-level techniques. The prior categories involve accessing application-level APIs on client devices. Network- and protocol-level techniques can also be used to fingerprint devices as follows.

(a) **IP address.** A client IP address can be used as an identifier, or to query regional Internet registries (via the WHOIS protocol) to obtain further information such as the Autonomous System (AS) it resides in and organization name it is registered to. While IP address is a more precise identifier than AS number, the latter is a more stable ID for hosts with dynamic IP addresses and is useful as a potential cross-check when verifying client location (see further comment regarding this in Section 5.6).

(b) **Geolocation.** The client's geographical location may be determined via several mechanisms. Modern web browsers typically expose APIs (e.g., via the navigator BOM object, cf. vector 1a) by which a website may request user permission to obtain current location (via, e.g., onboard GPS hardware, cellular triangulation, WiFi access point information, or user-supplied information). Network-based mechanisms [44] include WHOIS lookups based on IP address, inference based on routing data, and geolocation involving delay-based measurements [1].

(c) **Active TCP/IP stack fingerprinting.** Differences across network links and OS TCP/IP implementations allow devices to be fingerprinted by sending them carefully-crafted TCP/IP probes and analyzing response packet header fields (e.g., RTT, TCP initial window size) or link characteristics (e.g., MTU, round-trip delay). This method is browser-independent, and can be used on any Internet-accessible host. Nmap [37] is a popular scanning tool that includes host discovery, port scanning, and an OS detection feature that sends various probe packets and applies heuristics from its built-in database to differentiate thousands of systems. As this sends special probe packets to the client, i.e., is *active fingerprinting*, it can trigger firewall and IDS alerts due to its common reconnaissance use by attackers.

(d) **Passive TCP/IP stack fingerprinting.** A less intrusive (less powerful) technique, *passive fingerprinting*, sniffs existing network communication but uses heuristics similar to active fingerprinting to identify hosts; p0f [62] is an example of such a tool. Passive approaches are more suitable fingerprinting vectors, as header analysis of existing web traffic requires no new packets that may be flagged as intrusive.

(e) **Protocol fingerprinting.** Protocol-level fingerprinting can be applied to higher-level protocols to differentiate

[3]For example, on a system we tested, about 1ms elapsed while allocating each 16MB texture. This spiked to 10-40ms after the GPU VRAM reached full capacity.

[4]This error does not necessarily indicate all system memory is depleted, but that WebGL can no longer allocate memory. It is undesirable to continue allocating until this error triggers, as the user may be presented an error message (in our testing, desktop Google Chrome does not produce an error; Chrome for Android does).

versions or configurations of browser software or libraries. For example, the server may record HTTP header fields sent by clients, e.g., user-agent string, list of acceptable languages and character encodings, and the user-configurable DoNotTrack parameter. Moreover, the browser's TLS library can be fingerprinted using its ClientHello packet from the handshake sequence that negotiates protocol parameters [15]; information of relevance includes client TLS version, supported ciphersuites (and their order of presentation), compression options, and list of extensions (and associated parameters such as elliptic curve parameters).

(f) **DNS resolver.** A web server may determine which DNS resolver a client is using—for many clients, the default DNS resolver is configured by the user's ISP (which typically should not respond to queries originating from outside the ISP's network), but a minority of users may switch to other DNS resolvers, e.g., run by Google or OpenDNS. One approach involves a server sending the client browser a document containing a reference to a randomly-generated subdomain under a domain for which the authoritative DNS server is under control of the website owner [56]. When the client attempts to resolve the subdomain, the website's DNS server receives the request from the client's DNS resolver and can associate the randomly-generated subdomain with the client for which it was originally generated.

(g) **Clock skew.** TCP timestamps can be passively analyzed to measure client clock skew—a measure (e.g., in microseconds per second) of the rate at which the client's clock deviates from the true time [34].

(h) **Counting hosts behind NAT.** For clients behind a NAT, the number of hosts behind the NAT contributes to a device fingerprint. Bellovin [7] first proposed counting hosts behind a NAT by passively analyzing the IPv4 ID field (used for fragment reassembly); Kohno et al. [34] proposed the use of clock skew measurements to differentiate hosts behind a NAT. These techniques may be augmented with upper layer information, e.g., by including in the fingerprint the IDs of other users who access accounts from the same IP address.

(i) **Ad blocker detection.** While ad blocker detection can be done client-side with JavaScript as in vector 3(c), it can also be done server-side by monitoring incoming HTTP requests to detect if the client has requested the fake ad.

5. DEVICE FINGERPRINTING VECTORS: DESIRABLE PROPERTIES

Augmenting authentication with device fingerprinting requires considering various characteristics, which we now explore through a comparative analysis. While some characteristics may raise issues for behavioral advertising applications that use fingerprinting to track user browsing habits (not our focus), these may have less demanding requirements of device distinguishability; 80% or 90% accuracy in tracking user habits may suffice, but is rarely acceptable in authentication contexts (our focus). We discuss these properties below, with a comparative summary in Table 2.

To increase accuracy, device fingerprints employ multiple fingerprinting vectors. All vectors in Section 4 can be freely combined, limited by possible impact on device/user experience (only a few vectors, as noted below, have non-negligible resource costs), and server cost for fingerprint verification. Vectors from Categories 1 to 3 involve client-side JavaScript (and thus incur non-zero, but typically small, client overhead). To help compare vectors, and provide information of use in combining them, estimates of the distinguishability provided by each vector are included below and in Table 2.

5.1 Stability

Virtually all components of a device fingerprint are subject to change, but some (e.g., time zone for desktop machines or for mobile devices of users who rarely travel) may change much less frequently than others (e.g., browser version). A significant change in device fingerprint may require server authentication of the user to temporarily fall back to a less convenient but more reliable mechanism, such as a one-time passcode sent over SMS or e-mail. Thus it is preferable from a usability perspective to combine methods that provide fingerprints that are stable over time, i.e., with a sufficient number of component vectors stable at a given time. Overall fingerprint stability can be improved by combining multiple vectors and implementing a scoring mechanism that allows a subset of vectors to change. In the empirical study by Eckersley [21], an algorithm correctly linked a device's old fingerprint with the updated fingerprint in 65% of devices with 99.1% accuracy. If such an algorithm were used in the context of authentication, a high-security application might minimize false accepts at the expense of a higher false reject rate. This trade-off may be different in a context where security is less important than user convenience.

The majority of individual vectors from Categories 1 to 3 are relatively stable, with some exceptions:

- 1(a) Major software and hardware details may occasionally change if the user upgrades their OS, switches browsers, installs or removes browser plugins, etc.
- 1(d) Battery information will change, e.g., as battery capacity degrades with age.
- 2(a) HTML5 canvas rendering details may change with browser, OS, or graphics driver updates.
- 2(e) CSS feature detection and 2(f) JavaScript standards conformance may change with browser updates.

In contrast, vectors from Category 4 are generally less stable:

- 4(a) IP address, 4(b) Geolocation (depending on granularity), and 4(f) DNS resolver (unless manually configured) change as users log in from different locations. These vectors are more stable for desktop than mobile devices.
- 4(c, d) TCP/IP stack fingerprinting varies with routing changes that affect, e.g., round-trip delay and number of hops between client and server. If a user logs in from a different network location, additional changes would be observable in, e.g., the MTU of the network link, or responses to probe packets due to different firewall rules.
- 4(e) Protocol fingerprinting will vary with updates to the browser or shared (e.g., SSL/TLS) libraries.
- 4(h) Counting hosts behind a NAT is unstable over time, as devices may enter and leave a network.

Exceptions are 4(i) Ad blocker detection and 4(g) Clock skew (TCP timestamps reflect CPU clock skew, which is relatively stable [34]). Even vectors unstable over longer periods may remain useful for applications requiring identification across short periods, e.g., throughout one day, hour, or session.

		Citation	Stability	Repeatability	Low Resource Use	Spoofing Resistance	Client Passiveness	Distinguishing Info.	Overlaps With
Category 1: Browser-provided information	1a) Major software and hardware details	[21]	◓	●	●	○	○	○○○●○	1d; 2e,f,g; 3a,c; 4b,e
	1b) WebGL information	[61]	●	●	●	○	○	○●○○○ †	2a
	1c) System time and clock drift	[22]	●	●	●	○	○	○●○○○ †	
	1d) Battery information	[50]	◓	●	◓	○	○	○●○○○ †	1a
	1e) Evercookies	[2]	●	●	●	●	○	○○○○●	
	1f) WebRTC	[16]	●	●	●	○	○	○○●○○ †	
	1g) Password autofill	*	●	●	●	○	○	●○○○○	
Category 2: Inference based on device behaviour	2a) HTML5 canvas fingerprinting	[42]	◓	●	●	○	○	○○●○○	1b
	2b) System performance	[41]	●	◓	○	○	○	○●○○○ †	
	2c) Hardware sensors	[11, 19]	●	◓	◓	◓	○	○○●○○	
	2d) Scroll wheel fingerprinting	[49]	●	◓	●	○	○	●○○○○ †	
	2e) CSS feature detection	[58]	◓	●	●	○	○	●○○○○	1a
	2f) JavaScript standards conformance	[45]	◓	●	●	○	○	●○○○○	1a
	2g) URL scheme handlers	[30]	●	●	●	○	○	●○○○○	1a
	2h) Video RAM detection	*	●	◓	○	○	○	●●○○○ †	
	2i) Font detection	[48]	●	●	●	○	○	○●●○○	
	2j) Audio processing	[24]	●	●	●	○	○	○○●○○ †	
Category 3: Extensions and plugins	3a) Browser plugin fingerprinting	[21, 36]	●	●	●	○	○	○●●○○	1a
	3b) Browser extension fingerprinting	[41]	●	●	●	○	○	●●○○○	4i
	3c) System-fingerprinting plugins	[48]	●	●	●	◓	○	○○○●○	1a
Category 4: Network- and protocol-level techniques	4a) IP address	–	◓	●	●	●	●	○○○●○	
	4b) Geolocation	[44]	◓	●	●	◓	◓	○●●●○	1a
	4c) Active TCP/IP stack fingerprinting	–	◓	●	●	◓	◓	○●●○○ †	4d
	4d) Passive TCP/IP stack fingerprinting	–	◓	●	●	◓	●	○●○○○	4c
	4e) Protocol fingerprinting	[15]	◓	●	●	○	●	○○○●○ †	1a
	4f) DNS resolver	[16]	◓	◓	●	◓	◓	○●●○○	
	4g) Clock skew	[34]	●	◓	●	●	●	○○●○○	
	4h) Counting hosts behind NAT	[7]	○	◓	●	◓	●	○●○○○	
	4i) Ad blocker detection	–	●	●	●	○	●	●○○○○	3b

Table 2: Classification and comparative summary of device fingerprinting vectors. Citation sources provide discussion or original proposal; asterisk denotes unawareness of previous literature proposing the vector; dash indicates vector commonly known. Filled circle indicates that vector (row) provides benefit (column); half or empty circle indicates that benefit is partially provided or not provided, respectively. Distinguishing Info. uses a 5-dot scale: (VeryLow, Low, Medium, High, VeryHigh) with coarse mapping to bit-ranges (0-2, 3-6, 7-14, 15-30, ≥31); † denotes Requires Further Study (RFS). The final column notes vectors that overlap in the information that they provide.

5.2 Repeatability

We define repeatability as the property whereby a vector generates the same result if the software, hardware, and network configurations of a device are unchanged (whereas stability primarily concerns changes in device configuration). Repeatability is challenging for vectors measuring device performance, e.g., CPU, GPU, or network throughput, since performance varies if clients simultaneously perform other tasks. Most vectors discussed are repeatable, with these exceptions:

- 2(b) System performance may vary based on whether the device is burdened with other tasks. For mobile devices, it can also depend on temperature, as mobile chipsets scale back clockspeeds at high temperature.
- 2(c), 4(g) Temperature can affect hardware sensor data [19] and clock skew [46].
- 2(d) Scroll wheel fingerprinting requires that the user uses their scroll wheel. If the user does not do so on every visit to a web page, the vector is not repeatable.
- 2(h) Available VRAM will vary based on how much is currently in use by the device.
- 4(f) The DNS resolver used may vary due to load balancing of DNS resolvers.
- 4(h) Vectors counting hosts behind a NAT may vary depending on the presence of other devices on the network.

Fingerprinting vectors that are not reliably repeatable (but repeat often enough) may still be useful, if many vectors are used in the overall device fingerprint and an appropriate scoring mechanism is used as per Section 5.1, where a subset of vectors are allowed to change.

5.3 Resource Use (Overhead)

Fingerprinting vectors that require more system resources (e.g., CPU cycles, system memory, or I/O) bring performance and battery life costs. This is of less concern for websites that fingerprint devices only once per authentication process, and more for websites aiming to detect session hijacking by fingerprinting repeatedly throughout a session.

Most Table 2 vectors have low overhead, requiring processing time in the milliseconds range, but a few may require time on the order of seconds: 1(d) Battery information, 2(b) System performance, 2(c) Hardware sensors, and 2(h) Video RAM detection. While 2(b) consumes CPU cycles and 2(h) GPU memory, 1(d) and 2(c) only require time to collect sufficient data and thus rate half circles.

5.4 Spoofing Resistance

As per Section 3, attackers will try to spoof a device fingerprint that resembles the target. For vectors from Categories 1 to 3, the browser runs client-side JavaScript and returns output to the server; this makes it easy for attackers to spoof a (guessed or intercepted) response,[5] with three exceptions:

- 1(e) Evercookies are stateful (violating our pure fingerprinting definition), and can store global identifiers using, e.g., Flash cookies and HTML5 local storage, which are protected by Same-Origin Policy (SOP). Rather than *spoofing*, evercookies may be stolen and replayed by M5 attackers; but since this requires exploiting a vulnerability, we grant a filled circle.
- 2(c) While hardware sensors can be spoofed by obtaining a copy of the sensor data from the client and replaying it, obtaining such data in the first place may require user cooperation. For example, the web browser may ask for the user's permission before accessing the microphone.
- 3(c) Information available only through specialized plugins, e.g., hardware identifiers, may require more effort for attackers to obtain if the plugin is designed to communicate only with the website by which it was installed.

In comparison, Category 4 vectors are more resistant to spoofing (but none are completely immune). The following vectors are rated spoofing-resistant:

- 4(a) IP source address spoofing is possible due to insufficient deployment of source address validation across the Internet [10], and is often used to mount DoS attacks. However, using a spoofed source address to establish two-way communication with a host is difficult, as the host will always send response packets to the spoofed address.
- 4(g) Clock skew fingerprinting has been proposed for identifying wireless sensors in mesh networks [29] and access points in 802.11 wireless LANs [32]; both techniques rely on timestamps used in the respective MAC protocols. Arackaparambil et al. [4] showed that clock skew can be spoofed, but doing so introduces irregularities detectable through analysis, and that using a smaller 801.11 beacon frame transmission interval (typically 100ms) causes the irregularlities to be more pronounced and thus more easily detectable. Clock skew measurements derived from TCP timestamps would be more coarse-grained compared to MAC-protocol-level timestamps, but may nonetheless be difficult to spoof undetectably.

With the exception of 4(i) Ad blocker detection and 4(e) Protocol fingerprinting, which can be spoofed fairly easily by using the same browser and library versions as the target device, the remaining Category 4 vectors are graded as partially resistant to spoofing in Table 2:

- 4(b) Techniques like delay-based location verification [1] require attacker access to a proxy situated close to the victim to spoof the victim's location. Geolocation relying on information reported by a browser location API is more easily spoofed (but falls in Category 1).
- 4(c,d) Spoofing an OS TCP/IP stack can be done by manipulating TCP/IP behaviour with various off-the-shelf tools [9]. Network- or link-dependent measurements such as number of hops between client and server, round-trip delay, and MTU appear to require comparatively more effort to spoof.
- 4(f) If the client uses its ISP's DNS resolver (which typically responds only to DNS requests originating within the ISP's network), a successful attack requires access to a machine within that ISP's network.
- 4(h) The difficulty of spoofing the number of hosts behind a NAT varies based on the technique used; a simpler technique based on the IPv4 ID field [7] would be easier to spoof, whereas a clock skew based technique [34] would be more difficult (as discussed previously).

Resistance to spoofing may be increased by certain strategies. For example, with 2(a) HTML5 canvas fingerprinting, when a device is first associated with an account, the server may send multiple challenges (i.e., different sets of text and graphics to render), with the client returning corresponding results. The server may then randomly select a subset of stored challenges on each authentication attempt. Improving resistance of a vector to simple replay improves spoofing resistance, but a highly resourceful attacker might be able to configure attack software to mimic a target device for any given challenge, including for 2(a) as just noted. Different strategies may enhance the spoofing resistance of other vectors—e.g., for 2(b) System performance, a client puzzle [5] approach might be used, whereby the server sends cryptographic puzzles to the client and measures the time taken to receive a correct result. While more powerful devices might still spoof slower devices, such techniques raise the bar for attackers.

5.5 Client Passiveness

We classify a vector as client-passive if it can be used by the server without explicit cooperation or knowledge of the device. All vectors from Categories 1 to 3 require browsers to execute client-side JavaScript and return output to the server; these are not client-passive. All Category 4 vectors are client-passive to some degree, as none require explicit client cooperation. We elaborate in further comments.

- 4(b) Geolocation can be client-passive depending on mechanism, e.g., IP-based geolocation with table lookup is client-passive, but not vectors that require client-side JavaScript.
- 4(c) Active TCP/IP stack fingerprinting requires sending extra probe packets to the client to observe the response (or lack thereof). While sending probe packets does not require client cooperation, such packets may be detectable by client firewalls; probes crafted to appear part of regular HTTP traffic may be considered client-passive.
- 4(f) DNS resolver fingerprinting does not require client-side JavaScript, but on carefully inspecting webpage source code a client might suspect that the random-appearing domain name string was injected to learn the client's DNS resolver. Thus, we grade this vector partially client-passive.

[5] For example, credit card fraudsters use the FraudFox toolkit [26, 33], using a heavily modified Mozilla Firefox and Adobe Flash, to spoof various fingerprintable attributes such as OS version, screen resolution, and list of fonts. It includes a feature for capturing device fingerprints of phished users.

All others in Category 4 are graded client-passive; a device fingerprint can be constructed with these solely by inspecting existing HTTP traffic between server and client.

5.6 Distinguishing Information

To describe the granularity at which fingerprinting vectors can identify a device, we use the term *distinguishing information*; a related term is *personalization measurement* [24]. We prefer here to avoid directly using the term *entropy*, since, e.g., Shannon (and other) entropy is relative to an established probability distribution, which we are not in position to describe for device fingerprinting. In what follows we give only relative, qualitative estimates of distinguishability—informed in part by external studies. Distinguishability depends on the size of the (user) device space in question, and diversity of the underlying properties fingerprinted. A vector's distinguishability may depend on a site's target user distribution; e.g., for sites targeting a given country, vectors 1(c) System time and 4(b) Geolocation give less separation.

Given our informal definition of distinguishability, to help in comparing and selecting vectors to combine, we provide a qualitative analysis of each vector's distinguishability. Table 2 summarizes the ratings, and its caption gives the scale and coarse mapping for the estimates below; RFS denotes Requires Further Study.

- 1(a) Major software and hardware details is graded High, as it includes attributes such as user-agent string, list of plugins, screen resolution, and presence of localStorage and sessionStorage (˜10 bits, ˜15 bits, ˜4.83 bits, and ˜2.12 bits respectively as per [21]), in addition to other attributes.
- 1(b) WebGL information is estimated Low (RFS), based on our preliminary experimentation.
- 1(c) System time depends on time zone and daylight saving time; 24 possible values would yield a maximum of ˜4.65 bits, but in practice the distribution of users across time zones will be skewed, e.g., ˜3 bits as per [21]. Clock skew can also be measured, but this RFS, so we grade this vector overall as Low (RFS).
- 1(d) Battery information (RFS, but) is estimated Low for battery life, assuming that web-connected devices largely have battery life in the range of 4-10 hours of active use; using current charge level would provide more distinguishability for within-session fingerprinting.
- 1(e) Evercookies are graded VeryHigh; the server can save a globally unique identifier on the device.
- 1(f) WebRTC is graded Medium (RFS), as most consumer routers assign IP addresses in the 192.168.0.x range, thus these have upper bound ˜8 bits.
- 1(g) Password autofill and 4(i) Ad blocker detection are graded VeryLow; each are binary values.
- 2(a) HTML5 canvas fingerprinting is graded Medium, as it gives ˜8.6 bits of distinguishing information, per [36].
- 2(b) System performance (RFS, but) is estimated Low due to limited granularity at which system performance can be measured within the constraint of a few seconds.
- 2(c) Hardware sensors is graded Medium; Bojinov [11] collected accelerometer data from over 3000 devices, calculating the entropy of the distribution to be ˜7.5 bits.
- 2(d) Scroll wheel fingerprinting yields a binary value to distinguish scroll wheel from touchpad, but can also potentially detect differences in OS scroll speed settings; VeryLow (RFS).
- Vectors 2(e), 2(f), 2(g) are yet to be reported in empirical studies, but we estimate VeryLow; these vectors serve to distinguish between different browser vendors and versions, for which the distribution is likely skewed towards the most recent versions.
- 2(h) Video RAM is estimated VeryLow to Low (RFS). Many common configurations offer limited (say 2-3 bits of) choice, e.g., 1GB, 2GB, 4GB. (However this is complicated by many devices having shared system/video memory.)
- 2(i) Font detection is graded Low to Medium. Enumerating all fonts via the Flash plugin yields 7-14 bits [21, 36], with the spread largely attributable to substantially lower diversity of fonts on mobile devices. JavaScript font detection is unordered and cannot do a full enumeration (it tests using a list of known fonts).
- 2(j) Audio processing (RFS, but) is estimated Medium, since it is conceptually analogous to 2(a) [24].
- 3(a) Browser plugin fingerprinting includes full system font enumeration in addition to leakage of more granular system information (e.g., kernel version). However, mobile devices do not support plugins, and desktop web browsers are moving instead to the extension model [36], so we grade this Low to Medium.
- 3(b) Browser extension fingerprinting is graded VeryLow to Low, based on the expectation that few users install many browser extensions, aside from highly technical users. This may be too conservative, as some extensions (if installed) provide more distinguishing information, e.g., a NoScript whitelist.[6]
- 3(c) System-fingerprinting plugins can contribute considerable distinguishing information, since they have less restricted access to the underlying OS and hardware (see Section 4), compared to JavaScript. We grade it High.
- 4(a) IP addresses can often serve as global identifiers (but not always [17], e.g., due to NAT, proxying, and in some cases rapid address changes). We grade this vector High.
- 4(b) Geolocation can give considerable distinguishability, depending on the granularity of the specific geolocation mechanism. We grade this Low to High (high variability).
- 4(c,d) Based on the p0f [62] MTU and TCP flag signature lists, we estimate Low for 4(d) Passive TCP/IP stack fingerprinting, and Low to Medium for 4(c) Active TCP/IP stack fingerprinting; active probing is more powerful (RFS).
- 4(e) Protocol fingerprinting includes the list of HTTP headers (˜4.36 bits [36]), the values corresponding to certain headers such as user-agent string, HTTP accept headers (resp. ˜10 bits and ˜6 bits [21]), and DoNotTrack (up to 1 bit). It can also be inferred whether

[6] Published studies [21, 36], while skewed to technical users, did not specifically look at extensions beyond ad blockers. Technical users might improve privacy by, e.g., disabling Flash; using many browser extensions counters this.

cookies are enabled or disabled. SSL fingerprinting RFS. Thus we give an overall grade of High (RFS).

- 4(f) DNS resolver is analogous to geolocation, but less granular. We grade this Low to Medium.
- 4(g) Clock skew is graded Medium, as Kohno et al. [34] collected clock skew data for several thousand devices, calculating the entropy of the distribution to be ˜6 bits.
- 4(h) Counting hosts behind a NAT is graded Low; for household users, there is likely little variation in the number of hosts behind the NAT that will communicate with the web server, e.g., 1 to 16 devices (it would be higher for enterprise devices).

6. ROLE OF DEVICE FINGERPRINTING IN AUGMENTING AUTHENTICATION

To determine the most appropriate role for device fingerprinting in web authentication, we first discuss why it is unsuitable as a sole authentication mechanism. An idealized[7] form of device fingerprinting within the context of authentication might have the following properties:

P1: Each device has a unique fingerprint that can be associated with a user's account.

P2: Fingerprints obtained at different times from the same device are either identical; or linkable, i.e., can be determined with high confidence to be from the same device; or if changed to the extent of being unlinkable (e.g., due to major changes in software or hardware configuration), a backup mechanism such as e-mail or SMS-based recovery is in place to allow the user to re-associate a device with the target account.

P3: One of the following two properties is present:
  i) Fingerprints are released only to legitimate websites to which the user intends to authenticate.
  ii) It is difficult for an attacker, even with full knowledge of the device's hardware and software configuration, to spoof that device.

If the above requirements could be met, device fingerprinting alone could be used for account authentication. However, from Section 5, these requirements appear unreachable at present. It nonetheless remains possible to use device fingerprinting to strengthen authentication via an additional dimension (see Section 2), and importantly, without increasing the user burden—as opposed to, e.g., trying to improve password strength by forcing a heavier burden on users through a more complex password policy.

Device fingerprinting in this context has two use cases, pursued in subsections below: a) augmenting start-of-session authentication (cf. models M1-M4 of Section 3); and b) maintaining authentication throughout a session, to stop hijacking of authenticated sessions (related to model M5).

6.1 Authentication at Start of Session

When used as an added authentication dimension alongside passwords (or another primary authentication method), to authorize a session the server requires both the correct primary response and matching client fingerprint data. The server must thus have persistent access to data sufficient to verify the fingerprint on later authentication requests. The relevant time frame over which a fingerprint must be stable thus spans multiple sessions. This increases the importance of property P2 above—if the device configuration changes to the extent that an evolved fingerprint is no longer verifiable, a backup mechanism is needed to re-associate device and account.[8]

While resource use is relevant in the scenario of start-of-session authentication, a fingerprint need only be collected once at the start of the session; thus resource use is not a major barrier to arbitrary combinations of Table 2 vectors.

Augmenting two-factor authentication. Client fingerprinting can augment two-factor authentication ("factors" typically involve user actions; "dimensions" like device fingerprinting ideally do not). Consider Google two-step verification [28], which requires users to log in with a username and password, followed by a six-digit SMS code sent in real-time to a mobile device. If the user chooses to "trust" the computer on which they have logged in, a cookie is saved by the browser to relieve the user of entering a verification code on subsequent authentication attempts from that computer for the next 30 days. This is a security trade-off made in favour of usability, since an attacker obtaining the cookie bypasses the second factor. Device fingerprinting can be integrated into this scheme in at least three ways:

(a) When users submit a password, the server can validate the device fingerprint before sending the SMS code. If fingerprint verification fails, the server may require additional authentication tasks and/or send the user an alert—improving security in the event that the device used for receiving the SMS codes is stolen.

(b) When users submit a password, the server can require both a matching device fingerprint and a "trusted" cookie to bypass the second authentication factor (e.g., 6-digit SMS code)—improving security in case of cookie theft.

(c) Advertisers use device fingerprinting to restore tracking cookies after users clear them. This suggests a third application: if a user clears browser cookies and later attempts to re-authenticate, the server may recognize, by a device fingerprint, that the user previously designated the device as "trusted", and allow it to skip the second factor (i.e., authenticate by username-password alone). However, this approach appears to be a bad idea: users may clear their cookies precisely so that the server "forgets" their device for security reasons, e.g., on a device shared by multiple users.

6.2 Authentication Throughout a Session

In typical password-based web authentication (Section 6.1), upon receiving a username and password the server returns a browser session cookie, allowing the client to maintain its authenticated state by including the cookie in subsequent HTTP requests; the cookie, as a bearer token, replaces the password. Thus, an adversary obtaining the cookie (e.g., by device theft, cross-site scripting, interception due to a MITM attack or improperly configured HTTPS [35]) can submit authenticated requests without password knowledge.

Some websites record the user IP address when initiating an authenticated session, and check that any incoming HTTP requests containing a session cookie originate

[7]This characterization is impractical, but represents a highly favourable scenario for a web application wishing to accurately identify devices by fingerprints.

[8]Likewise if the user logs in from a different device.

from the same address [18]. Such address binding of session cookies enhances session security—if the address cross-check fails, the server can terminate the session, and optionally alert the user and lock down the account pending additional authentication steps. Since the client sends the session cookie alongside each HTTP request, the server ideally validates the source IP address on each request. This may impact usability, e.g., user IP addresses often change in mobile environments; here it may help to use multiple fingerprinting vectors.

Since modern webpages contain many resources, visiting a single page generates many HTTP requests; thus in this use case, client-passive vectors are critical. For *fully* client-passive vectors (including those in 4(b) that are client-passive, e.g., IP-based geolocation), the server can extract relevant fingerprint data from existing traffic flow, and can thus validate them on each HTTP request; this is not so for partially client-passive 4(c) or 4(f), which require extra network traffic.

Non-client-passive vectors require the browser to perform certain operations and generate fingerprint output for inclusion in each HTTP request. If the server uses fixed vectors, and which do not involve time-varying challenges, and the device configuration does not change, this fingerprint remains static for the duration of the user viewing a single webpage. Thus recomputing this fingerprint prior to each HTTP request is redundant for legitimate clients (and might impact user experience if vectors are high overhead); but, static fingerprints allow replay attacks by M5 attackers.

The SmartAuth framework [51] addresses replay attacks by hashing the device fingerprint at the client with a counter before sending to the server. This is ineffective against M5 attackers employing XSS to steal session cookies, as XSS can also steal the plaintext device fingerprint and counter. Such attackers can resume the session on their own machine and continue generating valid device fingerprints, e.g., incrementing the counter and recomputing the hash. An attack script may collect additional information about the target device to allow device spoofing in the event that the server dynamically adjusts its fingerprinting by, e.g., following the strategy employed by Unger et al. [58] of collecting a different subset of attributes each time the device is fingerprinted. M5, our most powerful attack model, grants session hijacking ability, and is more difficult to defend against.

In summary, for throughout-session fingerprinting:

1. Device fingerprints should be validated by the server for every HTTP request.
2. Advantages of client-passive vectors include (i) obscuring the server's fingerprinting strategy to the attacker;[9] and (ii) eliminating client resource burdens.
3. Periodically varying the fingerprinting challenge (see Section 3) improves spoofing resistance against attackers that intercept client-generated device fingerprints. However, forcing clients to regularly recompute device fingerprints increases resource usage costs.
4. Spoofing-resistance strategies that involve varying the format of client-generated device fingerprints (e.g., using counters and hashes, or varying the fingerprinted attributes used) can improve security but may be of limited effectiveness against advanced M5 attackers.
5. Vectors with higher spoofing resistance (see Table 2) provide stronger authentication assurances.

[9]While this temporarily increases spoofing resistance, recall our base assumption that this ultimately becomes known.

7. CONCLUDING REMARKS

Our classification and analysis is informed by previous studies as cited, and our own experimentation with 19 fingerprinting vectors. One conclusion from these is that combining essentially any subset of vectors in Table 2 appears feasible. Combining multiple vectors into a device fingerprint of course affects the properties discussed (see Sections 5.1 and 5.2). Distinguishability is expected to increase, but does not in all cases—e.g., if a smartphone's model number is in the user-agent string, screen resolution adds no further distinction, as all smartphones of one model have a given resolution. While an attacker may try to statistically guess some components of a device fingerprint—e.g., spoofing the most common screen resolutions—components such as hardware sensor calibration may be completely random and thus difficult to guess efficiently. While we cannot give precise quantitative guidance, combining more vectors tends to improve spoofing resistance, or at least raise the bar. Further exploration may involve a more advanced quantitative analysis of overall device fingerprint diversity and mechanisms for enabling users to associate multiple devices with their account.

We again emphasize that the fingerprinting mechanisms discussed herein require no new user interaction and thus impose no additional usability burdens on users; given increasing attention to usability, this strongly motivates the use of device fingerprinting to augment user authentication.

8. ACKNOWLEDGEMENTS

We thank Gunes Acar, Arvind Narayanan, Markus Duermuth, AbdelRahman Abdou, David Barrera, Daniel McCarney, and anonymous referees for helpful comments. The second author acknowledges NSERC funding under a Discovery Grant and as Canada Research Chair in Authentication and Computer Security.

9. REFERENCES

[1] AbdelRahman Abdou, Ashraf Matrawy, and P.C. van Oorschot. CPV: Delay-based location verification for the Internet. *IEEE Trans. Dependable Secure Comput.*, 2016. http://doi.ieeecomputersociety.org/10.1109/TDSC.2015.2451614.

[2] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proc. ACM CCS*, pages 674–689, 2014.

[3] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: Dusting the web for fingerprinters. In *Proc. ACM CCS*, pages 1129–1140, 2013.

[4] Chrisil Arackaparambil, Sergey Bratus, Anna Shubina, and David Kotz. On the reliability of wireless fingerprinting using clock skews. In *Proc. ACM WiSec*, 2010.

[5] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-resistant authentication with client puzzles. In *Security Protocols Workshop*, pages 170–177. Springer, 2000.

[6] Mika D. Ayenson, Dietrich James Wambach, Ashkan Soltani, Nathan Good, and Chris Jay Hoofnagle. Flash cookies and privacy II: Now with HTML5 and ETag respawning. *World Wide Web Internet and Web Information Systems*, 2011.

[7] Steven M. Bellovin. A technique for counting NATted hosts. In *Proc. ACM SIGCOMM Workshop on Internet Measurement*, pages 267–272. ACM, 2002.

[8] Adam Bergkvist, David C. Burnett, Cullen Jennings, Anant Narayanan, and Bernard Aboba. WebRTC 1.0: Real-time communication between browsers. https://www.w3.org/TR/webrtc/, 2016. Accessed: 2016-04-25.

[9] David Barroso Berrueta. A practical approach for defeating nmap OS-fingerprinting. https://nmap.org/misc/defeat-nmap-osdetect.html, 2003. Accessed: 2016-09-01.

[10] Robert Beverly, Ryan Koga, and K.C. Claffy. Initial longitudinal analysis of IP source spoofing capability on the Internet. ISOC whitepaper, 2013.

[11] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. Mobile device identification via sensor fingerprinting. In *arXiv preprint arXiv:1408.1416 [cs.CR]*, 2014.

[12] Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *Proc. IEEE Symp. Security & Privacy*, pages 538–552. IEEE, 2012.

[13] Joseph Bonneau, Cormac Herley, P.C. van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proc. IEEE Symp. Security & Privacy*, pages 553–567, 2012.

[14] Joseph Bonneau, Cormac Herley, P.C. van Oorschot, and Frank Stajano. Passwords and the evolution of imperfect authentication. *Communications of the ACM*, pages 78–87, July 2015.

[15] Lee Brotherston. Stealthier attacks and smarter defending with TLS fingerprinting. https://n0where.net/tls-fingerprinting/, 2015. Accessed: 2016-03-11.

[16] BrowserLeaks.com. https://www.browserleaks.com, 2011. Accessed: 2016-05-25.

[17] Martin Casado and Michael J. Freedman. Peering through the shroud: The effect of edge opacity on IP-based client identification. In *NSDI*, 2007.

[18] Gabriel Chen. Convenience over safety: How authentication cookies compromise user account security on the web. http://randomwalker.info/advising/undergraduate/chen-independent-work.pdf, 2014. Accessed: 2016-05-01.

[19] Anupam Das, Nikita Borisov, and Matthew Caesar. Tracking mobile users through motion sensors: Attacks and defenses. In *Proc. NDSS*, 2016.

[20] Michael Dietz, Alexei Czeskis, Dirk Balfanz, and Dan S. Wallach. Origin-bound certificates: A fresh approach to strong client authentication for the web. In *Proc. USENIX Security Symp.*, 2012.

[21] Peter Eckersley. How unique is your web browser? In *Proc. 10th Int. Conf. Privacy Enhancing Technologies*, pages 1–18, 2010.

[22] Ori Eisen. Method and system for identifying users and detecting fraud by use of the Internet. US Patent, December 2010. US 7853533 B2.

[23] Ori Eisen. Systems and methods for detection of session tampering and fraud prevention. US Patent, April 2012. US 8151327 B2.

[24] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. http://randomwalker.info/publications/OpenWPM_1_million_site_tracking_measurement.pdf, 2016. Draft: May 18, 2016.

[25] David Fifield and Serge Egelman. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security*, volume 8975 of *LNCS*, pages 107–124. Springer, 2015.

[26] FraudFox. http://www.wickybay.xyz/2016/01/fraudfox.html, 2016. Accessed: 2016-05-25.

[27] David Mandell Freeman, Markus Dürmuth, and Battista Biggio. Who are you? A statistical approach to measuring user authenticity. In *Proc. NDSS*, 2016.

[28] Eric Grosse and Mayank Upadhyay. Authentication at scale. *IEEE Security & Privacy*, 11(1):15–22, January 2013.

[29] Ding-Jie Huang, Wei-Chung Teng, Chih-Yuan Wang, Hsuan-Yu Huang, and Joseph M. Hellerstein. Clock skew based node identification in wireless sensor networks. In *Proc. IEEE GLOBECOM*, 2008.

[30] IT Security Solutions. Fingerprinting browsers using protocol handlers. http://itsecuritysolutions.org/2010-03-29_fingerprinting_browsers_using_protocol_handlers/, 2010. Accessed: 2016-04-25.

[31] Markus Jakobsson, Elaine Shi, Philippe Golle, and Richard Chow. Implicit authentication for mobile devices. In *Proc. USENIX HotSec*, 2009.

[32] Suman Jana and Sneha K. Kasera. On fast and accurate detection of unauthorized wireless access points using clock skews. *IEEE Trans. Mobile Comput.*, 9(3):449–462, March 2010.

[33] Jeremy Kirk. This tool makes it easier for thieves to empty bank accounts. PCWorld: http://www.pcworld.com/article/2872372/this-tool-may-make-it-easier-for-thieves-to-empty-bank-accounts.html, 2015. Accessed: 2016-05-25.

[34] Tadayoshi Kohno, Andre Broido, and K.C. Claffy. Remote physical device fingerprinting. *IEEE Trans. Dependable Secure Comput.*, 2(2):93–108, April 2005.

[35] Michael Kranch and Joseph Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *Proc. NDSS*, 2015.

[36] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *Proc. IEEE Symp. Security & Privacy*, 2016.

[37] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[38] Maxmind Developer Site. Device tracking add-on for minFraud and proxy detection services. http://dev.maxmind.com/minfraud/device/. Accessed: 2015-11-17.

[39] Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. In *Proc. IEEE Symp. Security & Privacy*, pages 413–427, 2012.

[40] Aleecia M. McDonald and Lorrie Faith Cranor. A survey of the use of Adobe Flash local shared objects to respawn HTTP cookies. *ISJLP*, 7:639–687, 2011.

[41] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting information in JavaScript implementations. In *Proc. Web 2.0 Security & Privacy*, 2011.

[42] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Proc. Web 2.0 Security & Privacy*, 2012.

[43] Mozilla Developer Network. WebGL. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API. Accessed: 2015-11-23.

[44] James A. Muir and P.C. van Oorschot. Internet geolocation: Evasion and counterevasion. *ACM Comput. Surv.*, 42(1):4:1–4:23, 2009.

[45] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, and Edgar Weippl. Fast and reliable browser identification with JavaScript engine fingerprinting. In *Proc. Web 2.0 Security & Privacy*, 2013.

[46] Steven J. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proc. ACM CCS*, pages 27–36, 2006.

[47] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. PriVaricator: Deceiving fingerprints with little white lies. In *Proc. WWW*, 2015.

[48] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proc. IEEE Symp. Security Privacy*, pages 541–555, 2013.

[49] Jose Carlos Norte. Advanced Tor browser fingerprinting. http://jcarlosnorte.com/security/2016/03/06/advanced-tor-browser-fingerprinting.html, March 2016. Accessed: 2016-03-11.

[50] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. The leaking battery: A privacy analysis of the HTML5 Battery Status API. *IACR Cryptology ePrint Archive*, 2015.

[51] Davy Preuveneers and Wouter Joosen. SmartAuth: Dynamic context fingerprinting for continuous user authentication. In *Proc. ACM SAC*, pages 2185–2191, 2015.

[52] Daniel Roesler. STUN IP address requests for WebRTC. https://github.com/diafygi/webrtc-ips, 2015. Accessed: 2016-04-25.

[53] Suphannee Sivakorn, Iasonas Polakis, and Angelos D. Keromytis. The cracked cookie jar: HTTP cookie hijacking and the exposure of private information. In *Proc. IEEE Symp. Security & Privacy*, 2016.

[54] Ashkan Soltani, Shannon Canty, Quentin Mayo, Lauren Thomas, and Chris Jay Hoofnagle. Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management*, 2010.

[55] Jan Spooren, Davy Preuveneers, and Wouter Joosen. Mobile device fingerprinting considered harmful for risk-based authentication. In *Proc. European Workshop on System Security*, 2015.

[56] Stack Overflow. Is it possible to detect visitor DNS server? http://stackoverflow.com/questions/10721731/is-it-possible-to-detect-visitor-dns-server. Accessed: 2015-10-27.

[57] The 41st Parameter. The 41st parameter announces new real-time product as world's first standard for online PC identification. http://www.the41.com/buzz/announcements/41st-parameter-announces-new-real-time-product-worlds-first-standard-online-pc-0, July 2006. Accessed: 2015-11-17.

[58] Thomas Unger, Martin Mulazzani, Dominik Fruhwirt, Markus Huber, Sebastian Schrittwieser, and Edgar Weippl. SHPF: Enhancing HTTP(S) session security with browser fingerprinting. In *Proc. 8th Int. Conf. Availability, Reliability and Security*, pages 255–261, September 2013.

[59] Tom Van Goethem, Wout Scheepers, Davy Preuveneers, and Wouter Joosen. Accelerometer-based device fingerprinting for multi-factor mobile authentication. In *Engineering Secure Software and Systems*, volume 9639 of *LNCS*, pages 106–121. Springer, 2016.

[60] Thomas Emmanual Varghese, Jon Bryan Fisher, Steven Lucas Harris, and Don Bosco Durai. System and method for fraud monitoring, detection, and tiered user authentication. US Patent, March 2011. US 7908645 B2.

[61] Valentin Vasilyev. fingerprintjs2. https://github.com/Valve/fingerprintjs2, 2015. Accessed: 2016-05-25.

[62] Michal Zalewski. p0f v3. http://lcamtuf.coredump.cx/p0f3/. Accessed: 2015-10-26.