4-1-2023

Mitigation of JavaScript-Based Fingerprinting Attacks Reliant on Client Data Generation

Nathan Joslin
University of Dayton

Mitigation of JavaScript-Based Fingerprinting Attacks Reliant on Client Data Generation

Honors Thesis

Nathan Joslin

Department: Computer Science

Primary Advisor: Phu H. Phung, Ph.D.

Secondary Advisor: Ahmed El Ouadrhiri, Ph.D.

April 2023

Mitigation of JavaScript-Based Fingerprinting Attacks Reliant on Client Data Generation

Honors Thesis

Nathan Joslin
Department: Computer Science
Primary Advisor: Phu H. Phung, Ph.D.
Secondary Advisor: Ahmed El Ouadrhiri, Ph.D.

April 2023

Abstract

While fraud detection companies use fingerprinting methods as a secondary form of identification, attackers can exploit these fingerprinting methods due to the revealing nature of the software and hardware information collected. Attackers can use this sensitive information to target users with known vulnerabilities, monitor a user's activity, and even reveal their identity without their knowledge or consent. Unfortunately, average users have limited options to opt out of or block fingerprinting attacks.

In this thesis, we propose a solution that enforces dynamic policies on web pages to prevent potential malicious device fingerprinting methods. We employed the Inline Reference Monitor (IRM) approach to supervising JavaScript operations on web pages, including method calls, object creation and access, and property access. When executed, the IRM will intercept these operations, providing runtime policy enforcement to mitigate JavaScript-based dynamic fingerprinting methods that generate unique data at runtime instead of collecting static attributes. In particular, our policy enforces a randomization method rather than normalization or domain-based blocking to constantly change a given device's fingerprint over time, making it increasingly difficult for malicious actors to track a device across the web. Our approach can protect user privacy while limiting major site breakage, a common issue with current anti-fingerprinting technologies.

We have performed intensive experiments to demonstrate the effectiveness of our approach. In particular, we replicated and revised an existing fingerprinting attack that collects network link-state information to construct unique fingerprints. We deployed this fingerprinting attack on the cloud and collected data from web users nationwide, which are used by a machine learning model to reveal users' locations with high accuracy. We have implemented our mitigation method by extending a browser extension prototype. The prototype demonstrated that our proposed method could effectively prevent data collection from the fingerprinting attack.

Acknowledgements

I'd like to thank Dr. Phung, Dr. Ahmed, and Ms. Paola for advising me throughout the project, my family for their support, and any participants in the data collection for this project.

University *of*
Dayton

Table of Contents

Abstract **Title Page**

1 Introduction 1

2 Background 3

2.1 Overview of Fingerprinting Methods 3

2.2 Entropy and Stability of Fingerprint Features 6

2.3 Challenge/Response-based Authentication.................................. 8

2.4 Mitigation Approaches 10

3 Problem Description 12

3.1 The PingLoc Prototype 13

3.2 Proposed Solution... 14

4 Implementation 16

4.1 Reproducing the PingLoc Web Application 16

4.2 Data Collection and Processing 18

4.3 Feature Extraction... 22

4.4 Machine Learning.. 24

5 Results 25

5.1 User Localization Through Machine-Learning Classification ... 25

5.2 Mitigation with Code-Origin Policy Enforcement 25

6 Limitations 28

7 Future Works 31

References 32

Chapter 1: Introduction

Digital fingerprinting is the technique of collecting attributes associated with a browser or device. These attributes are combined to form a unique identifier, a fingerprint, which may be used for stateless tracking or identification/recognition purposes [4]. The fingerprint of one device or browser will differ from another as a result of variations in software and hardware configuration. Fingerprinting methods are used in industry and research as a layer of fraud protection and as a means of additional authentication in a multi-factor authentication scheme [12].

Although fingerprinting has increased security in the fraud detection sector, it also poses many challenges to the everyday web user as a stateless tracking mechanism [21]. Users cannot simply delete their fingerprints as they can with cookies. As a result, a third-party can collect device fingerprints to track user activity across the web without the user being aware of it or being able to stop it [8]. However, the application of browser fingerprinting is not limited to unique user identification and tracking. Eckersley et al. claim that even users who are not uniquely identifiable are still vulnerable to more context-specific fingerprinting attacks [4]. We consider this involuntary tracking and collection of private data to violate user privacy, and thus malicious by nature. Other concerns raised by fingerprinting include the revealing of sensitive information, the ability to re-spawn cookies [5], and even linking fingerprints to social media accounts to reveal user identities [8].

The first main contribution of this work includes a reproduction of the PingLoc prototype developed by Wu et al. [20]. PingLoc is one of the first link-based fingerprinting techniques, implementing a multilateration cross-site image request scheme to generate them. The core idea behind this scheme is to reveal the location of users by using ‘distances’ to known locations. Instead of using direct distances, this scheme uses a sequence of request time-delay values to multiple known servers to derive network links’ information. By collecting the time-delays of a multitude of image requests to various geographically diverse servers, PingLoc can extract statistical features that reflect the link-state information of the network a device is connected to. These features are subsequently used to train a machine learning model to localize users, which achieves up to 98.5% accuracy according to Wu et al. [20].

The second main contribution of this work includes a proposal for a few improvements to the implemented multilateration cross-site image resource request scheme [20]. The most significant of these improvements is a proposal of four additional statistical features extracted from the time delay data, namely interquartile range, interquartile first quarter, interquartile third quarter, and the number of lost packets. We theorize that these additional features will increase the accuracy of the machine-learning model.

Finally, the third main contribution of this work is a security policy that successfully blocks the multilateration cross-site image resource request scheme proposed by [20]. Our security policy is enforced by the Inline Reference Monitor (IRM) developed by Phung et al. [15]. Their policy enforcement mechanism allows for code-origin policy enforcement, which provides a means for developers to enforce more flexible security policies. Using this policy enforcement mechanism, we develop a code-origin policy that prevents link-based fingerprinting through image requests. Although our policy only blocks link-based fingerprinting that uses *image* requests to gather link-state information, it can easily be modified to handle other object requests.

Chapter 2: Background

It’s important to have a clear understanding of browser/device fingerprinting before discussing the particular method reproduced in this work. Classification as well as a brief overview of the types of fingerprints collected by fingerprinting algorithms helps bring to light how these methods work, what information they collect, and why certain countermeasure approaches are chosen. These fingerprinting techniques are often split into two groups: passive and active. Passive fingerprinting is the collection of device or browser attributes via HTTP headers, such as the user-agent string or language. Active fingerprinting is the collection of attributes via execution of client-side JavaScript, such as installed fonts, screen resolution, or plugins [16]. The vast majority of browser/device fingerprinting methods are classified as active, and thus heavily dependent on JavaScript function and method calls.

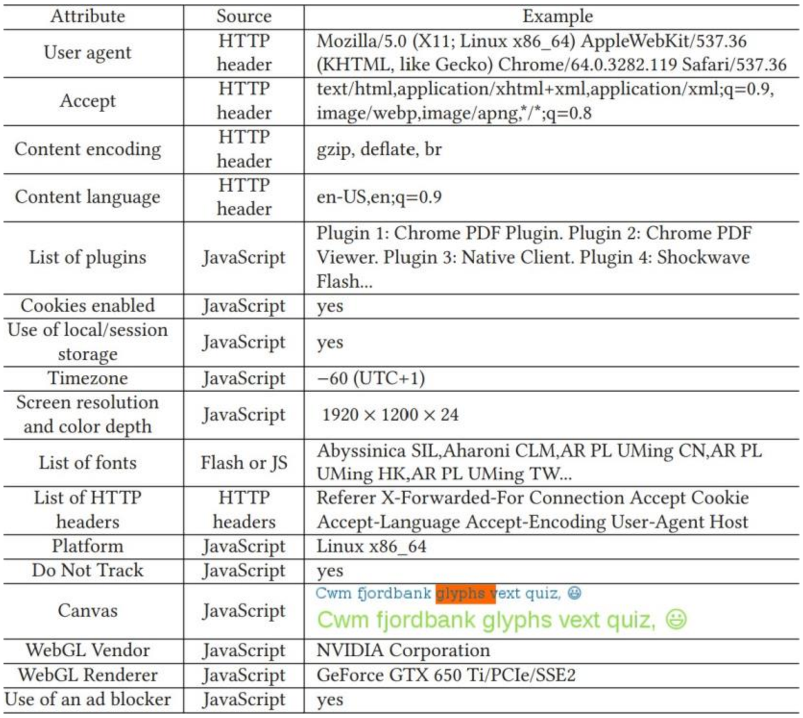
2.1 Overview of Fingerprinting Methods

The root of browser fingerprinting research rests in the hands of Peter Eckersley [4]. Eckersley was the first to propose the concept of browser fingerprinting. Since then, numerous browser and device fingerprinting methods have been researched. In an attempt to provide an understanding of the different types of fingerprinting methods, we will briefly introduce a few of the most prevalent in literature.

Browser Fingerprints

The majority of browser fingerprinters are JavaScript object-based. As the name indicates, these techniques gather a variety of attributes from JavaScript objects, exploiting differences in user browser configurations. The most common objects used by these techniques are navigator objects, which contain browser information, and screen objects, which contain display configuration. Collecting a breadth of these attributes allows the building of a unique fingerprint [21, 8]. Figure 2.1, from [10], is an example of a browser fingerprint.

However, some browser fingerprinting techniques do not rely on JavaScript APIs at all. A novel example is the CSS-based fingerprinting technique StylisticFP developed by Xu Lin et al. [14] which relies exclusively on CSS features. CSS-based fingerprinters rely on parsing CSS, often to identify the presence of particular plugins or extensions [21, 11]. These fingerprinters are particularly dangerous as the prevalence of certain extensions may reveal sensitive information about the user whose browser is being fingerprinted [18, 7]. For example, detecting the presence of an accessibility extension or an extension focused on a particular religion.



Attribute	Source	Example
User agent	HTTP header	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.119 Safari/537.36
Accept	HTTP header	text/html,application/xhtml+xml,application/xml;q=0.9, image/webp,image/apng,*/*;q=0.8
Content encoding	HTTP header	gzip, deflate, br
Content language	HTTP header	en-US,en;q=0.9
List of plugins	JavaScript	Plugin 1: Chrome PDF Plugin. Plugin 2: Chrome PDF Viewer. Plugin 3: Native Client. Plugin 4: Shockwave Flash...
Cookies enabled	JavaScript	yes
Use of local/session storage	JavaScript	yes
Timezone	JavaScript	−60 (UTC+1)
Screen resolution and color depth	JavaScript	1920 × 1200 × 24
List of fonts	Flash or JS	Abyssinica SIL,Aharoni CLM,AR PL UMing CN,AR PL UMing HK,AR PL UMing TW...
List of HTTP headers	HTTP headers	Referer X-Forwarded-For Connection Accept Cookie Accept-Language Accept-Encoding User-Agent Host
Platform	JavaScript	Linux x86_64
Do Not Track	JavaScript	yes
Canvas	JavaScript	Cwm fjordbank glyphs vext quiz, 😃 Cwm fjordbank glyphs vext quiz, 😃
WebGL Vendor	JavaScript	NVIDIA Corporation
WebGL Renderer	JavaScript	GeForce GTX 650 Ti/PCIe/SSE2
Use of an ad blocker	JavaScript	yes

Figure 2.1: An example of a browser fingerprint[10].

Device Fingerprints

The distinction between browser and device fingerprinting in literature is relatively vague. Both fingerprinting methods ultimately rely on the browser as an ‘attack vector’. Intuitively, a browser fingerprinter extracts browser-specific information, while a device fingerprinter extracts device information. Where this distinction becomes unclear is in the user-agent string, where some OS information is extracted but as a JavaScript call to the browser. Largely, device fingerprinters may be distinguished by their dynamic approach. Instead of making a few JavaScript object property accesses and storing the aggregate information, they generate a different form of data such as a processed audio signal or a rendered image.

A few hardware and software-based fingerprinting methods are Canvas, AudioContext, and WebGL. Contrary to browser fingerprinters, these fingerprinters exploit the differences in the hardware and software running on a device [21]. The scripts run by these types of fingerprinters generate unique data by tasking a client to render images, process audio signals, or generate another form of data. For example, a canvas fingerprinter creates an image using JavaScript method calls. This is typically done by drawing a pangram on a canvas element as well as introducing more complex curves, color gradients, or shadows to yield a more detailed image [12]. Figure 2.2, created by [12], is an example of a rendered canvas image by acanvas fingerprinter. The canvas element's data is then collected and condensed using the convenient method toDataURL provided by the canvas API, which is the fingerprint itself. For canvas and WebGL fingerprinters, the resulting data is unique enough to serve as the fingerprint itself in some cases [12]. However, this is not the case for all data-generating fingerprinting methods such as AudioContext [17].

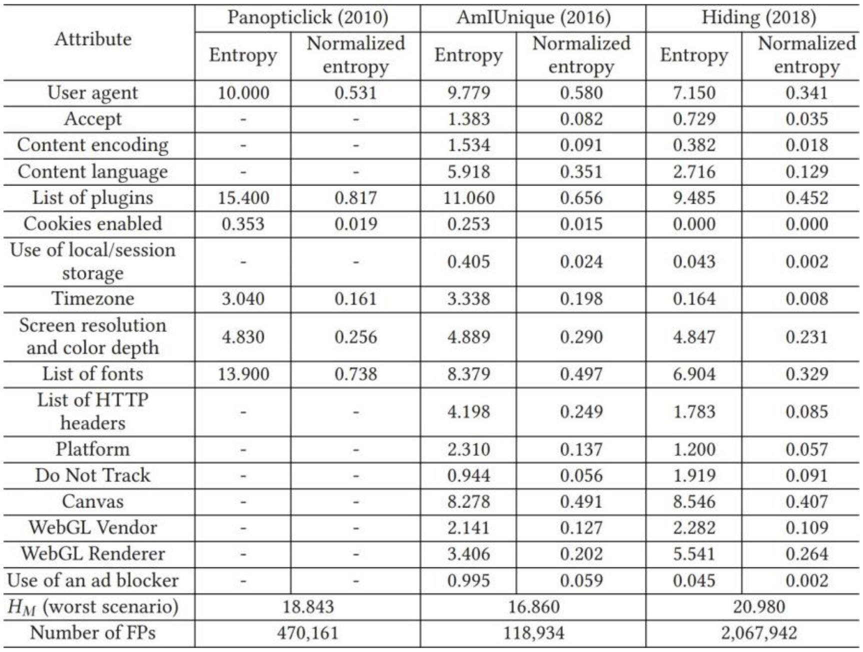
Figure 2.2: An example of a rendered canvas image by a fingerprinter [12].

2.2 Entropy and Stability of Fingerprint Features

A fingerprinting feature is only as valuable as the information entropy it yields. In [4], Eckersley provides a method of representing fingerprinting features mathematically. Although a mathematical analysis is beyond the scope of this work, a high-level overview of the information entropy and stability of fingerprinting features helps to complete our understanding of the current literature.

In [4], Eckersley provides a crucial insight concerning the relationship among features. In some instances, fingerprinting features are independent of one another. For example, there is no correlation between browser type and language used. On the other hand, there is a correlation between OS and screen resolution; a mobile OS cannot have a desktop screen resolution. As the majority of fingerprinting algorithms rely on the aggregation of multiple features to construct unique fingerprints, it is crucial to consider these types of relationships among features.

A few works have studied the information entropy of fingerprinting features through empirical analysis. Laperdrix et al. [10] provide an excellent table demonstrating the uniqueness of individual attributes. This table, see Figure 2.3, is a collection of data from three prior studies Panopticlick, AmIUnique, and Hiding. All attributes listed on the left are static attributes, except for Canvas which is dynamic. It is quite apparent that some static attributes provide more uniqueness, or entropy, resulting from the attribute's possible range of values.



Attribute	Panopticlick (2010)		AmIUnique (2016)		Hiding (2018)	
	Entropy	Normalized entropy	Entropy	Normalized entropy	Entropy	Normalized entropy
User agent	10.000	0.531	9.779	0.580	7.150	0.341
Accept	-	-	1.383	0.082	0.729	0.035
Content encoding	-	-	1.534	0.091	0.382	0.018
Content language	-	-	5.918	0.351	2.716	0.129
List of plugins	15.400	0.817	11.060	0.656	9.485	0.452
Cookies enabled	0.353	0.019	0.253	0.015	0.000	0.000
Use of local/session storage	-	-	0.405	0.024	0.043	0.002
Timezone	3.040	0.161	3.338	0.198	0.164	0.008
Screen resolution and color depth	4.830	0.256	4.889	0.290	4.847	0.231
List of fonts	13.900	0.738	8.379	0.497	6.904	0.329
List of HTTP headers	-	-	4.198	0.249	1.783	0.085
Platform	-	-	2.310	0.137	1.200	0.057
Do Not Track	-	-	0.944	0.056	1.919	0.091
Canvas	-	-	8.278	0.491	8.546	0.407
WebGL Vendor	-	-	2.141	0.127	2.282	0.109
WebGL Renderer	-	-	3.406	0.202	5.541	0.264
Use of an ad blocker	-	-	0.995	0.059	0.045	0.002
H_M (worst scenario)	18.843		16.860		20.980	
Number of FPs	470,161		118,934		2,067,942	

Figure 2.3: A table of browser attributes and their respective entropy [10].

Furthermore, it is important to note the formal definitions used by Pugliese et al. where a fingerprint is *stable* if, and only if, its stability period is greater than zero; a notably loose definition as the authors do not specify a unit of time [16]. Consequently, a fingerprint is *trackable* if, and only if, the fingerprint is unique to one user and is stable [16]. However, the necessity for a fingerprint to be unique to one particular user depends on the goal of the fingerprinting algorithm. For instance, the goal of a fraud detection company that aims to authorize a set of users is fundamentally different when compared to that of a fingerprinter which aims to identify vulnerable software running on a device. Differentiating between each user is necessary for the former to create a secure authentication protocol, but not necessary for the latter as they only care about identifying the presence of a particular unit of software. Nevertheless, the goal of the majority of fingerprinting algorithms is to differentiate between users.

With rapid software development comes unavoidable, frequent, device and browser re-configuration. It is imperative to consider the stability of a given fingerprint over time as attributes evolve. The causes for a change in a device fingerprint may be classified into three categories: Browser or OS Updates; User Actions, such as changing time zone; and Environment Updates, e.g., updating browser-related software such as emojis or installing a new plugin [13]. The three causes mentioned result in changes of static attributes, e.g., an OS update results in a change of the user-agent string provided by a browser.

As a result of the stability limits of fingerprinting features, it is important to consider the research done to track an evolving fingerprint over long periods. Many studies are not broad enough, focusing on one particular method of fingerprinting which poses limits on the application of the proposed method. One of the best studies done thus far was performed by Pugliese et al. [16] who conducted a three-year study on browser fingerprinting. Their study provides vital information concerning the formalization of fingerprinting concepts, the uniqueness, and stability of fingerprints, as well as data on the relation between user characteristics, e.g., age or education, and trackability. Ultimately, they conclude that desktop fingerprints are stable for 11.6 weeks on average and 89.2% of desktop fingerprints are trackable, i.e. the new fingerprint is linkable to the prior. However, it is important to note that in prior works, such as FP-Stalker [19], the overall effectiveness decreases as the number of users within the fingerprint database increases. This is also discussed by [13], who claim that once a dataset of fingerprints reaches the millions the time consumption for uniquely identifying a fingerprint begins to become a problem. However, smaller studies show that it is possible to create an algorithm to track evolving fingerprints [21].

2.3 Challenge/Response-based Authentication

Although one of the core goals of this work is to design and implement mitigation policies for device fingerprinters, fingerprinting is used positively where mitigation may not be desired. These fingerprinting applications are used to verify the identity of a user rather than to exploit the user's sensitive information or track them. As mentioned previously, the best example of this non-malicious use of fingerprinting is fraud detection as an additional means of authentication. In [12], Laperdrix et al. proposes that a challenge/response-based authentication mechanism may be created by utilizing active fingerprinting techniques. In this work, they use canvas fingerprinting to create the authentication protocol, however, they emphasize that other dynamic feature-generating fingerprinting techniques may be used as well, such as WebGL and AudioContext. The proposed protocol, see Figure 2.4, works after the user visits the site once, i.e., the user can be authenticated from their second visit to the web page and any time after [12]. When visiting the web page, the client is given a canvas "challenge" to render. The newly rendered canvas image is then compared to the same image generated during the prior visit. If these images match exactly, then the client is authenticated. Following the verification process, the client is given a second canvas challenge to render. This challenge will be used to authenticate the user during their next visit to the web page.

It is important to note that protocols such as this one are effective until a change in system hardware or software is made. For instance, an organic GPU driver update may alter how the canvas image is rendered. As a result, this mechanism by itself is not reliable enough and other means of multi-factor authentication are necessary to fall back on. One solution to this lack of fingerprint stability is SMS or another means of authentication. Following the fallback mechanism, the previous canvas fingerprint for a client may be overwritten with the new one. Other solutions involve algorithms and ML models that link an unrecognized fingerprint to a previous fingerprint by keeping track of these predictable organic changes in hardware and software, such as clustering algorithms.

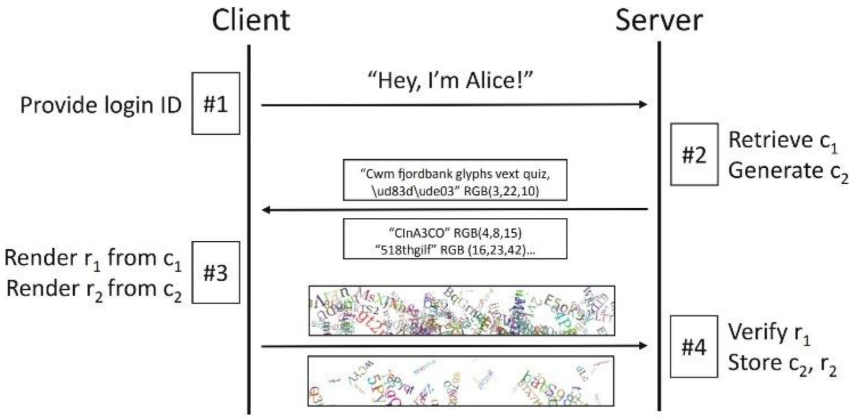


Figure 2.4: A challenge/response-based authentication protocol using canvas fingerprinting [12].

2.4 Mitigation Approaches

Anti-Fingerprinting Privacy Enhancing Technologies (AFPETs) typically follow a few approaches. First, the normalization technique takes a "hide in the crowd" approach. Also known as 'attribute standardizing' [2], this method aims to reduce the entropy of fingerprints by setting attributes to default values. A few examples of normalized values are the user-agent string, time zone, and screen resolution. However, this technique requires a sufficient user base to successfully create a "crowd" to hide in. Furthermore, if a user of a normalized fingerprint were to change a single attribute value they would deviate from the crowd. Thus, according to Laperdrix et al., a user would be easily fingerprintable and may be more easily done than without the use of the normalization technique. Despite these limitations, this method is actively used by the Tor browser.

A secondary technique used by AFPETs is randomization. Also known as 'attribute varying' [2], these methods are similar to normalization ones. Instead of "hiding in the crowd", randomization techniques aim to create a "moving target". Rather than spoofing similar values for a given user base to reduce uniqueness, this technique will regularly change device attributes or introduce noise to generated features to change the collected fingerprint. By doing so, a browser fingerprint is constantly changing, making it increasingly difficult for a fingerprinter to track a user over time or across the web. Fingerprinting browser feature research has proven that randomization is an effective countermeasure to canvas fingerprinters [12]. To demonstrate this, we built our own canvas fingerprint poisoner which can be found in our GitHub repository: https://github.com/isseclab-udayton/MyWebGuard-AntiFingerprinting.

The third technique used by AFPETs involves blocking mechanisms. Also known as 'interaction blocking' by [2], these methods block the execution of a particular API or interactions with particular domains (block-listing). While some techniques involve complete API blocking, such as the Tor browser blocking the rendering of all canvas elements by default as a countermeasure to canvas fingerprinters. Other techniques involve partial or temporal API blocking, such as those based on identifying patterns in a web page's behavior like Ad-Graph [6].

Chapter 3: Problem Description

In this work, we attempted to reproduce the fingerprinting prototype PingLoc. Designed by Wu et al. in [20], this technique leverages the link-state information of a device to use as fingerprinting features. Despite what one might think, the authors find that the link-state information is distinct and stable. As a result, they can extract statistical features from a device’s link-state information to form a "physical location fingerprint" [20]. Using link-state information as a fingerprinting feature is arguably more invasive to user privacy than other features, despite yielding less entropy. While link-based fingerprinting may not be able to uniquely identify users, it is able to reveal information more sensitive than the majority of information collected by other methods, geolocation.

The proposed multilateration cross-site image resource request scheme is not restricted by the Same-Origin-Policy (SOP), Cross-Origin Resource Sharing (CORS), or the X-Frame-Options HTTP header. According to the MDN Web Docs[3], CORS prevents images from third-party origins from being loaded into canvas elements. As an HTTP-header-based mechanism, CORS is a setting that lets servers indicate other origins that may load the requested resource. A closer examination of the types of requests affected by the CORS header reveals that only images drawn to canvas elements can be blocked with this header. By loading the third-party images into iframes, the multilateration image request scheme is able to bypass SOP and CORS. As a result, this bypassing of policies can be viewed as a form of leakage of device link-state information.

Furthermore, the X-Frame-Options HTTP response header, which prevents browsers from loading particular pages into iframes or other objects, does not affect the prototype. Largely, the X-Frame-Options header is used by developers to prevent their web pages from being embedded into malicious ones. With this single header developers can add a layer of security to their web page, further preventing it from being included in click-jacking attacks. Although the proposed scheme requests images and not web pages, the X-Frame-Options header does not affect it because the ping time can still be parsed from the file path included in the error. Although the original authors do not mention the X-Frame-Options header [20], we feel it is worth mentioning as it directly pertains to iframes. Nevertheless, the implemented multilateration cross-site image resource request scheme is robust against these existing security mechanisms, and a new security approach is necessary for the everyday web user to maintain control over their privacy.

3.1 The PingLoc Prototype

In [20], Wu et al. "propose a multilateration cross-site image resource request scheme to obtain the physical location fingerprint of users according to time delay." The scheme sends image requests, or *pings*, to a set of geographically diverse servers, collecting the time delay as the images return. More specifically, they add the current date and time, when the ping is being sent, to the end of the image path. As a result of the path concatenation, the image request will most certainly return an error. After parsing the returned error for the start time inside the file path, they calculate the round-trip time for the image request based on the received time. These pings are sent out to a set of servers every 800ms and collected over a window of time. Thus, the resulting raw data is a series of windows of request time delays organized by the destination server.

The link-state information is then derived from each server window by calculating its statistical features. Seven statistical features are extracted from each server window of data collection: Maximum, Minimum, Mean, Variance, Root-Mean-Square, Skew, and Kurtosis. The first four of these features are relatively intuitive, however, the final three deserve more attention. The Root-Mean-Square of a window reflects the noise of the data window [20]. The Skew reflects the skew direction and degree of data distribution [20]. Finally, the Kurtosis of a window reflects the steepness of the data distribution [20]. The result of feature extraction is a two-dimensional array of the seven features, where again sub-arrays represent different pinged servers. This two-dimensional array is then combined to form a single array of features by concatenating the sub-arrays.

Finally, the authors process the features using a min-max standardization method; a method commonly applied to data used in machine learning applications. By fitting the extracted features into the interval of [0,1], the differing magnitude of feature values has less effect on subsequent model training. The resulting trained model is able to localize user browsers [20].

3.2 Proposed Solution

The conflicting uses of browser fingerprinting by benign and malicious actors create a paradox. On one hand, many organizations and corporations are trusted by users. For example, a banking institution utilizing browser fingerprinting as a means of challenge-response based authentication may be accepted by users as it provides them with another layer of security over their sensitive information [12]. At the same time, a user may not want a third-party executing fingerprinting scripts through ads and tracking them as they browse the web. However, a mitigation approach that completely blocks fingerprinting methods would disrupt both benign and malicious cases. The question we aim to answer is: How can we allow organizations we trust to fingerprint our browsers, while at the same time blocking it from those we don’t trust?

In this work, we leverage the MyWebGuard policy enforcement mechanism developed by Phung et al. [15] to enforce security policies on webpages that engage in dynamic browser fingerprinting, i.e., fingerprinting methods that require the client to generate unique data. MyWebGuard is implemented as a browser extension, and functions as an Inline Reference Monitor (IRM) that intercepts the JavaScript operations executed on webpages. Three types of JavaScript operations can be monitored: method calls, object creation, and property access. The IRM intercepts the execution of these operations, allowing for security policy enforcement prior to the intended action. As this policy enforcement mechanism monitors JavaScript operations, it makes an excellent method to mitigate JavaScript-based fingerprinting attacks.

The Same Origin Policy (SOP) prevents JavaScript code from external origins from accessing internal origin data. However, if an origin includes third-party code the said code is treated as if it belongs to the first-party origin. A core mechanism of the MyWebGuard framework is its ability to distinguish the origins of JavaScript code. This mechanism allows security policies enforced by the MyWebGuard IRM to provide a more flexible, fine-grained configuration. In this way, MyWebGuard provides an extension to the SOP which Phung et al. refer to as *Code Origin Policy* [15]. By enforcing code-origin policies with MyWebGuard based on block lists, we can permit fingerprinting to be used for challenge/response-based authentication from trusted domains while still maintaining privacy where it matters to the user [12].

Chapter 4: Implementation

4.1 Reproducing the PingLoc Web Application

We implement a Go HTTP server deployed on Heroku to serve the static JS, HTML, and CSS files. The server also receives the collected data from users, storing the information in a MongoDB collection. In our reproduction of the PingLoc web application prototype, we made a few modifications to how the data is collected. Figure 4.1 shows the web app architecture and may be accessed on Heroku: https://mywebguard-antifingerprinting.herokuapp.com/. Additionally, the web application can be found on our Github: https://github.com/isseclab-udayton/MyWebGuard-AntiFingerprinting/tree/main/testing/web_app.

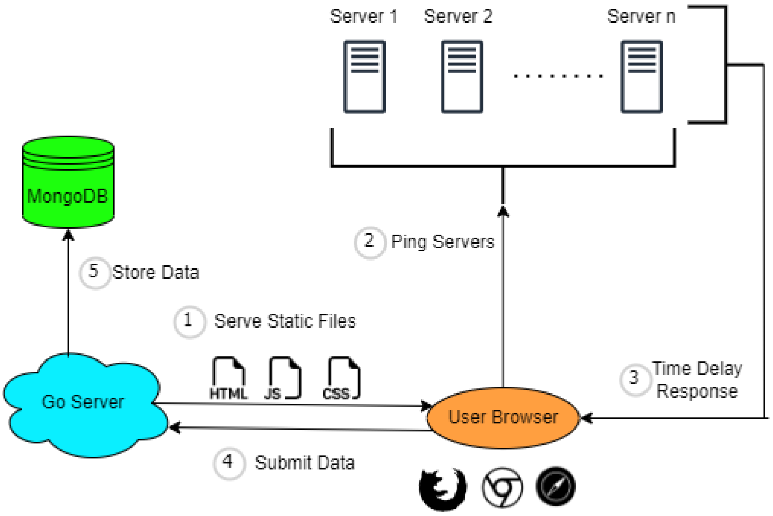


Figure 4.1: Architecture of the data collection web application.

For our first modification, we sent the image requests to a different set of servers. In the original implementation, Wu et al. [20] set a threshold of 800ms, at which point the request packet is assumed to have been lost, to improve sampling efficiency. As we experienced a far larger packet loss at this threshold than the 3% experienced by the original authors, we hypothesized that the distance to the original servers was too far and a more localized set of servers was necessary for more reliable data. Even if we're mistaken and the distance to the servers was not the culprit, the large amount of packet loss would have made data collection and feature extraction much more difficult. Thus, we chose a more localized set of servers. More specifically, we chose to use eleven university servers from across the United States. Table 4.1 provides more information about the servers used in our reproduction of the PingLoc prototype.

Our second modification to the data collection process is in how the requests are sent to the servers. Instead of requesting by domain name, e.g., *stanford.edu* as done by Wu et al. [20], our image requests are sent directly using the IP address of each server. The presence of replicated content delivery networks (CDNs) is the motive behind this change. Wu et al. use domains such as *baidu.com, iqiyi.com,* and *douban.com* in their prototype implementation. These domains belong to sizeable companies somewhat comparable to those of *google.com* or *facebook.com* as the majority of these domains are in the Alexa top 100. Large companies like these are known for their use of CDNs. By sending image requests to these domains, DNS servers may route different clients to different servers based on geolocation. For instance, a client on the east coast of the U.S. pinging an east-coast *facebook.com* server may yield similar time delays to a west-coast client pinging a west-coast server. Events such as these may have a significant impact on the accuracy of the machine-learning model.

For our server selection process, we first chose eleven geographically diverse universities. Each university website homepage was visited and the domains were used in *nslookup* commands to obtain the IP addresses. We put each IP address into various IP geolocation websites for additional verification that the server is in the intended geographic area. If the consensus of the IP geolocation websites suggested that the server may not be on or near the university campus, we assume some sort of hosting service was being used and a new university was chosen. Thus, by sending the image requests to universities, as opposed to industry, and by using an IP address rather than the domain we are confident that a particular server will be reached by all clients. A summary of the servers used in our reproduction can be found in Table 4.1.

Finally, the minor modifications we made include the following: a cloud-based deployment; the use of HTTPS, rather than HTTP; and data visualization tools used during and after the collection process.

Servers Used in Reproduction			
University	State	Domain	IP Address
Stanford	California	stanford.edu	171.67.215.200
Oregon State	Oregon	oregonstate.edu	52.27.33.250
Auburn	Alabama	auburn.edu	131.204.138.170
Alaska Fairbanks	Alaska	uaf.edu	137.229.114.150
Texas A&M	Texas	tamu.edu	165.91.22.70
Penn State	Pennsylvania	psu.edu	128.118.142.114
North Dakota U.	North Dakota	und.edu	134.129.183.70
Colorado College	Colorado	coloradocollege.edu	198.59.3.123
Maine	Maine	umain.edu	130.111.46.127
Wisconsin	Wisconsin	wisc.edu	144.92.9.70
Florida State	Florida	fsu.edu	146.201.111.62

Table 4.1: The servers used in our reproduction of the PingLoc prototype.

4.2 Data Collection and Processing

Collecting data for a multilateration cross-site image resource request scheme is a challenging problem. A geographically diverse set of user devices is necessary for reliable data. Outsourcing the data collection process to businesses such as Amazon's AWS Device Farm may have been possible if a budget was available. Another option was to create the data ourselves using a set of Virtual Machines and a Virtual Private Network to make it appear to the network as if the devices were in other locations. However, this second option was not chosen as the use of VMs or VPNs may have had a significant impact on the reliability of the data collected. Thus, we attempted to contact real users by sending out emails requesting help with the process. Emails were sent to students, coworkers, friends, or family; strongly encouraging them to forward it to others. Furthermore, posts were made on LinkedIn to encourage those in related academia or industry to participate.

As mentioned by Wu et al. [20], the collected time delays must go through data processing and feature extraction before the data can be used in the testing and training of a machine learning model. A complex network environment is bound to experience congestion, packet loss, and other phenomena that may introduce noise to the collected data. As a result, the raw time delay data itself does not provide the information necessary for training a classification model. Thus, in preprocessing the data by extracting statistical features we can reduce the noise experienced by the model.

We first removed entire instances of data collection in quite a few cases. In 15 of these cases, the value of the city field, given by the user, was either null or "City". These instances of data are unusable as the location of the user was unknown, amounting to 21% of the dataset. In other cases, a few select instances were excluded as the user experienced too much packet loss and thus statistical features could not be extracted following their removal. The cause of these large amounts of packet loss is largely attributed to the user browsing other tabs during the data collection process. Furthermore, we excluded one instance where nearly every time delay value was over 800ms, which should be impossible in theory as the timeout threshold was set to 800ms; see Figure 4.2. Finally, we exclude the cities with only one instance of data collection, as it is difficult to draw a conclusion on the network of a city based on one user. Table 4.2 provides an overview of the dataset, where only cities with more than one instance are included. The remaining cities with only one instance are combined into the ‘Other’ category.

Database Overview	
City Field Value	Instances
"null"	12
"City"	3
"Dayton"	6
"Columbus"	6
"Liberty Township"	4
"Los Angeles"	2
"Framingham"	2
"Wildwood"	2
"Menomonie"	2
"Minneapolis"	2
Other	29

Table 4.2: An overview of our collected data. The Other category is every city with only a single instance of data collected.

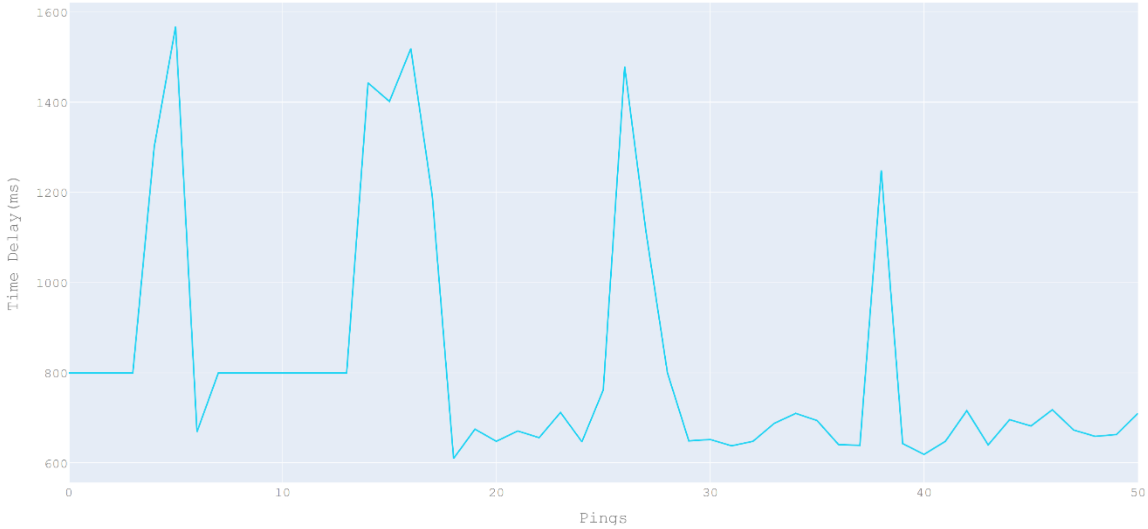


Figure 4.2: A removed instance containing invalid data. A trace of time delay values from a client to a single server.

Following the exclusion of the invalid instances, the remaining dataset was insufficient to train a machine-learning model. To obtain a sufficiently large dataset we duplicated the valid raw data, introducing a small amount of noise. Our process for creating synthetic data is as follows. For each server window in an instance of valid data, we build a synthetic instance by first generating an array of random values between zero and one. This array is then multiplied by the noise value *n* and added to the original window, effectively introducing anywhere between zero and *n* milliseconds of noise per time delay value. Generating synthetic data in this way most definitely has its faults, which are discussed later in the Limitations section. Although this is not ideal for training a machine learning model, the generation of synthetic data was necessary to complete this work.

After creating synthetic data based on the real dataset, we then selected windows from each server trace in an instance. Wu et al. [20] discuss the importance of selecting an appropriate window size, which is subsequently used in feature extraction. Intuitively, they claim if the selected window is too small, then the statistical features of the data are not properly reflected. Furthermore, if the selected window is too large, then the large amount of similar neutral data frames generated will also affect the training result. From experimentation, they suggested a window size between 20 and 30 to obtain the greatest model accuracy. Figure 4.4 demonstrates the synthetic traces derived from the real data in Figure 4.3.

Although using a more local set of servers should cause significantly less packet loss and reduced propagation delays, some packet loss still occurs. In our implementation, we use the same time delay threshold of 800ms. As a result, packets lost or packets with longer than 800ms propagation delay have their time delays set to the threshold. As suggested by [20], we remove all time delays greater than or equal to 800ms prior to feature extraction.

4.3 Feature Extraction

Wu et al. [20] select several statistical features to extract from each window. As such, in each instance of data collected we extracted features for each server trace, i.e., each of the eleven traces of pings included in an instance is reduced in size to a 20–30-point subset. The features concerning each window are Maximum, Minimum, Mean, Variance, Root-Mean-Square, Skew, and Kurtosis. As the values of each statistical feature vary, a min-max standardization is performed before model training. Each respective feature for each server is standardized, i.e., for all data instances used to train a model the means of server n's pings are standardized together as opposed to all means among all servers being standardized together.

We implement the same methods of feature extraction and min-max standardization. However, we also include our own set of statistical features in an attempt to improve model accuracy. We propose adding four new features to the original set, increasing the total number of features extracted per server from seven to eleven. These additional features include the number of lost packets, interquartile range, interquartile first quarter (Q1), and interquartile third quarter (Q3) and were inspired by [1].

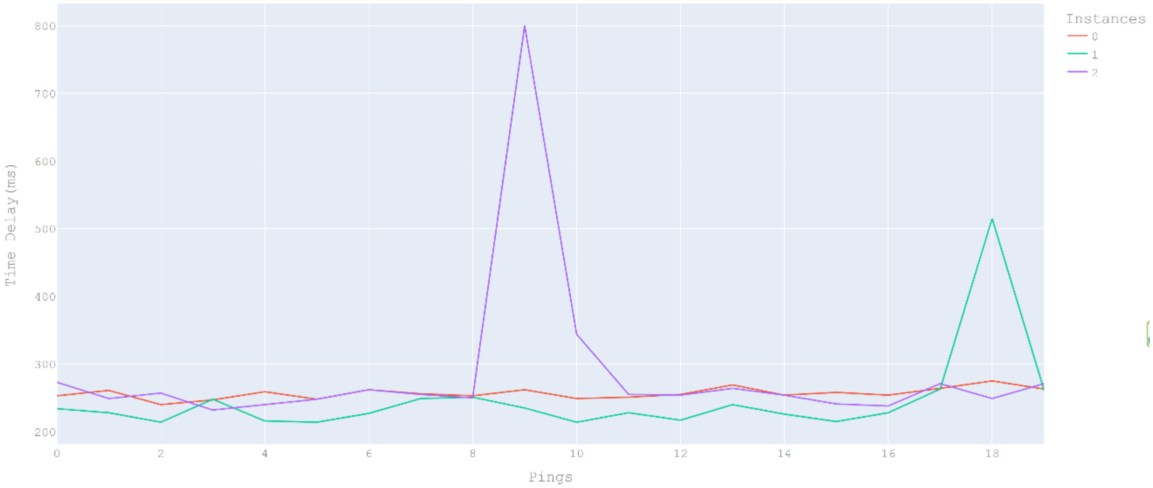


Figure 4.3: A graph of the real traces of three Columbus users' pings to one server.

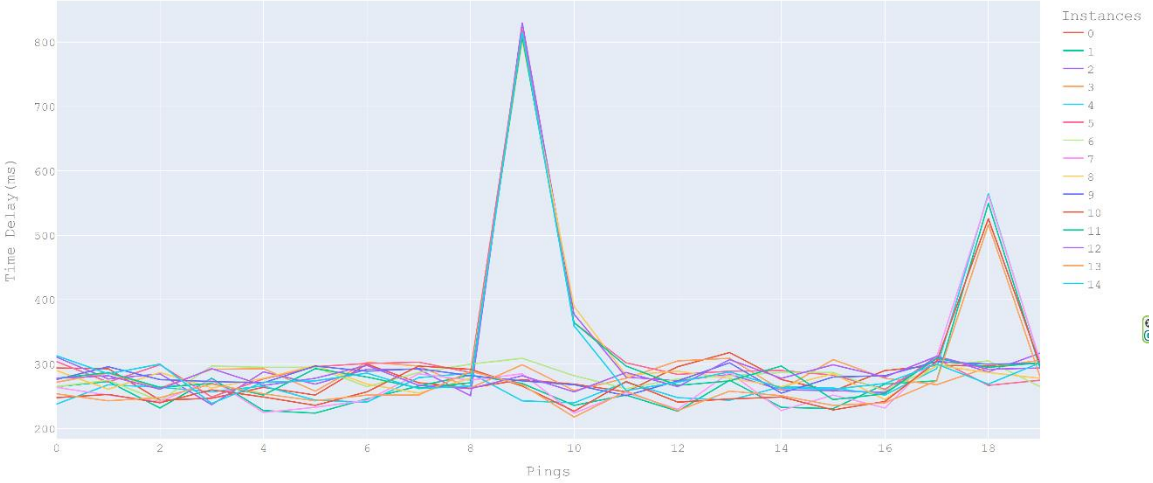


Figure 4.4: A graph of the synthetic traces created from the real traces in Figure 4.3.

The first of these proposed features is a count of the number of packets lost. Although it is important to remove these time delays before feature extraction, disregarding them completely does not accurately reflect the network link state information as these dropped packets may indicate network congestion. Although the number of lost packets is low overall, i.e., the majority of values for this feature remains at zero, if a particular infrastructure link experiences frequent packet loss, then the cities utilizing this link would be more easily distinguishable in theory.

The other three of the proposed features were inspired by [1] who use electrocardiogram signals to identify healthcare patients by extracting statistical features and training a Random Forest model, achieving greater than 99% accuracy. Throughout the data collection process, we noticed that many traces of pings followed an interesting pattern, see Figure 4.5. In some cases, the network would be stable with the time delay from one ping to the next fluctuating within a tight range, e.g., the time delays remain between 60ms and 80ms consistently. However, during what appeared to be times of network congestion, the time delays would reach similar maximums, e.g., the congested pings have a typical range of 150ms and 200ms. Similar behavior can be seen in an ECG trace where the high voltage points sensed by leads remain consistent. Although the rhythm of the human heart is far more consistent and stable than that of a network, we find that similarities may still be found in both traces. If this network behavior occurs, it can be captured by the interquartile range, Q1, and Q3 features.

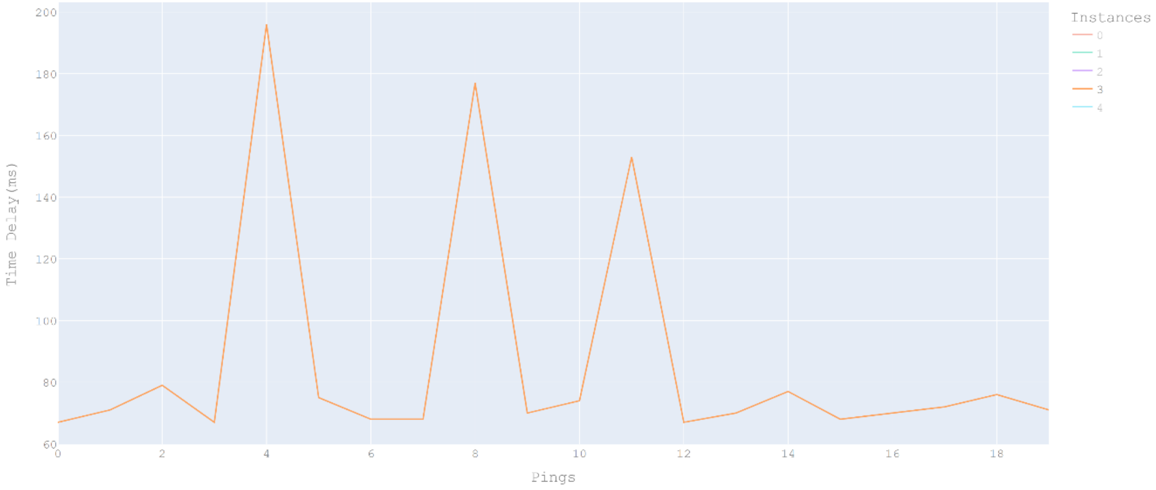


Figure 4.5: A trace of pings of a client to one server. The interquartile range, Q1, and Q3 features are intended to capture the behavior of traces such as this one.

4.4 Machine Learning

For user classification/identification, we chose to implement a K-Nearest-Neighbors (KNN) algorithm as this was the most accurate for Wu et al. [20]. The KNN algorithm assumes that similar data points appear near each other when graphed. The algorithm classifies a new data point based on the existing *K* data points nearby. The resulting prediction is then determined by a majority vote. In our case, we assume that the link-state information is similar for users geographically close to one another, e.g. users within the same city. The resulting features extracted from the synthetic data were split into testing and training data. We used 25% of the feature vectors for testing and the remaining for training. We trained two machine learning models both with a K value of 45; one for the original feature extraction method (Figure 5.1) and another including our suggested features (Figure 5.2).

Chapter 5: Results

5.1 User Localization Through Machine-Learning Classification

By using the feature extraction method proposed by Wu et al. [20], a K value of 45, and introducing a noise between 0 and 25ms to the real data when creating synthetic data we were able to obtain an accuracy of around 91.2%; see figure 5.1. This is notably less than the 98.5% achieved by [20] which is discussed in the limitations section.

By using our feature extraction method, a K value of 45, and introducing a noise between 0 and 25ms to the real data when creating synthetic data we were able to obtain an accuracy of around 84.8%; see Figure 5.2. Thus, it appears our feature extraction method was not as accurate as the original proposed by Wu et al. in [20]. Although our model is less accurate than the one produced by Wu et al., the resulting sets of training data look quite similar when projected into two-dimensional space, see Figures 5.1 and 5.2. This suggests that the errors seen in the model are likely due to the noise introduced to the data. Indeed, the noise we introduced interrupts the trends intended to be captured by our proposed statistical features. As a result, we are inclined to believe that our data set, in its limited size, does not accurately reflect real-world data.

5.2 Mitigation with Code-Origin Policy Enforcement

The code-origin-based policies proposed in this work are enforced by blocking the loading of images. The time delay information cannot be collected if the image response is never fully received. As a result of the blocking policy, every ping reaches the timeout threshold. The collected data is then not usable to localize the client. Figure 5.3 shows the link-based fingerprinting attack in action, and Figure 5.4 shows the result of activating the MyWebGuard extension. As seen by the figures, the mitigation method successfully blocks any link-state information from being collected.

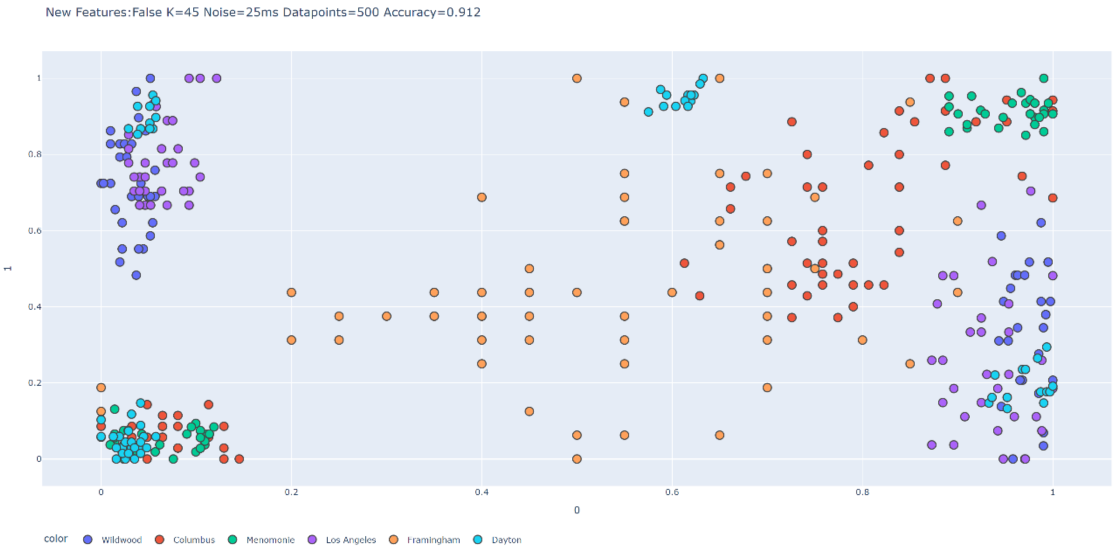


Figure 5.1: This graph represents the training data obtained by using the original feature extraction method. The features include the window: maximum, minimum, mean, variance, root-mean-square, skew, and kurtosis. Different colors represent different cities, see legend bottom left.

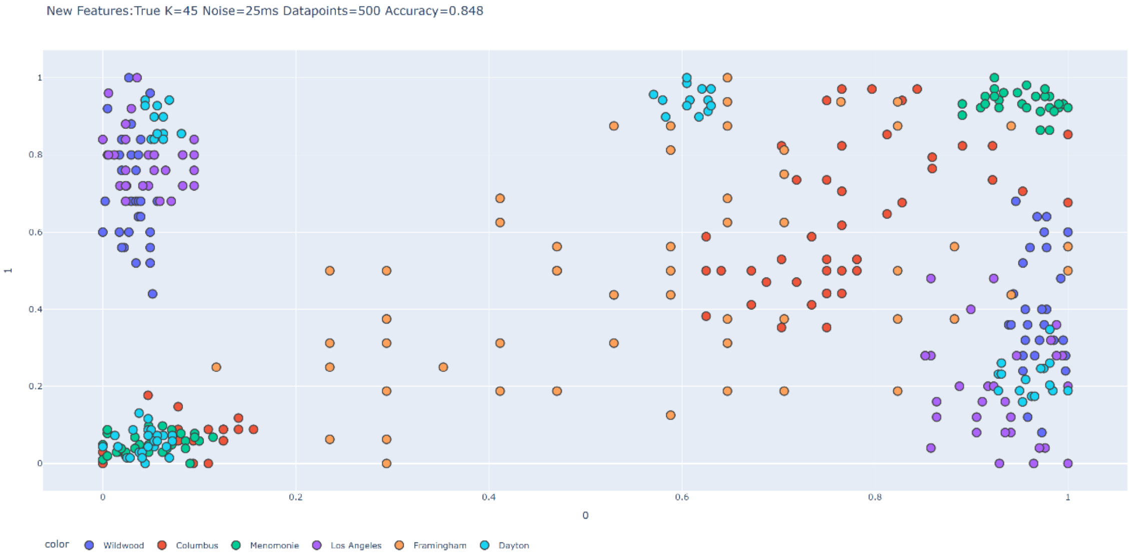


Figure 5.2: This graph represents the training data obtained by using our feature extraction method. The features include the window: maximum, minimum, mean, variance, root-mean-square, skew, kurtosis, interquartile range, Q1, Q3, and packets lost. Different colors represent different cities, see legend bottom left.

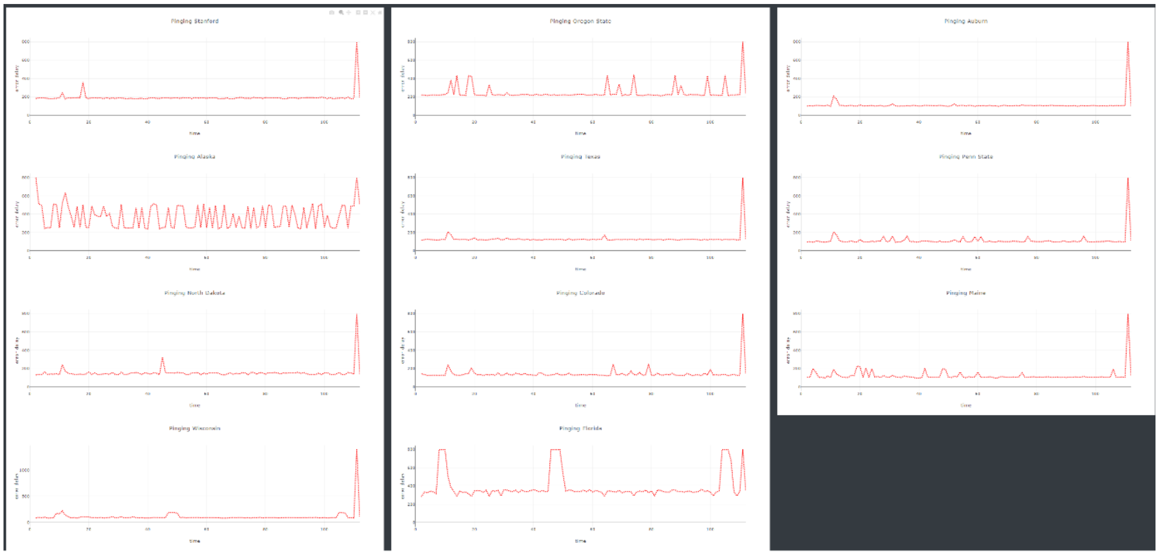


Figure 5.3: The traces of time delays for every server used in the data collection process. The MyWebGuard extension is **not** active. Screenshot from https://mywebguard-antifingerprinting.herokuapp.com/.

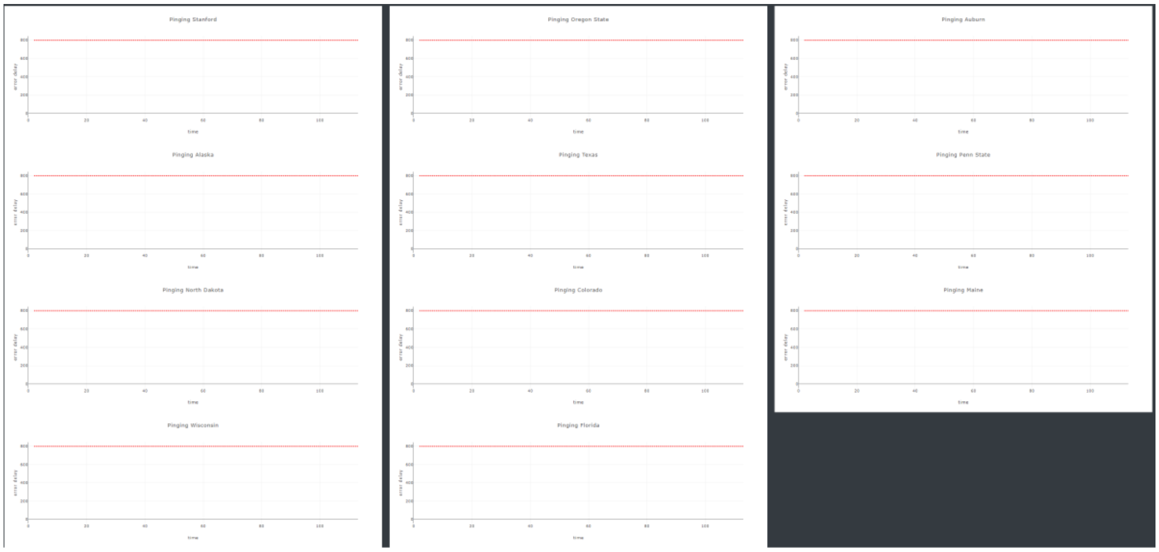


Figure 5.4: The traces of time delays for every server used in the data collection process. The MyWebGuard extension is active. As seen, the fingerprinting method is ineffective. Screenshot from https://mywebguard-antifingerprinting.herokuapp.com/.

Chapter 6: Limitations

First, the classification done by the PingLoc prototype [20] only reveals the geolocation of a user. As such it cannot distinguish different users at the same location, let alone users within the same city. Said differently, using link-state information as a fingerprinting feature does not yield enough entropy to uniquely identify devices. Despite these limitations, link-based device fingerprinting remains a novel technique as it provides a new vector for gathering client geolocation data.

The second limitation of the PingLoc prototype is its reliance on bursts of HTTP requests. The large number of requests generated is bound to create network congestion at larger scales. Consequently, the networks would begin behaving differently as they handle the increased demand. The resulting changes in the links of the network would also change the fingerprints, rendering all prior fingerprints invalid.

As for our reproduction of the PingLoc prototype, one possible limitation could be the use of university servers rather than large businesses with CDNs. These businesses use CDNs to provide fast and reliable content to clients. Using large business servers rather than university servers could result in more stable network connections. Consequently, the link-state information extracted from the time delay data may be more effective model training data. Furthermore, we had trouble collecting a sufficient amount of data to properly train a machine learning algorithm. We duplicated the valid data we had and introduced noise to form synthetic traces. However, the randomization method we used to create the noise significantly impacts the statistical features extracted from them. As a result, the synthetic traces used for model training do not accurately reflect real-world data, see Figures 6.1 and 6.2.

As mentioned previously and according to Laperdrix et al. [12], dynamic feature-generating fingerprinting techniques may be used in challenge/response-based authentication protocols. As an active fingerprinting feature reliant on client-generated data, the link-based technique shares similarities with the methods discussed by [12]. Thus, the PingLoc prototype may arguably be used as a challenge/response-based authentication protocol. However, one major aspect of link-based device fingerprinting that is not shared by the other active fingerprinting techniques is the time it takes to generate a sufficient amount of information to construct the fingerprint. In [12], canvas fingerprinting is an excellent example of an active technique capable of challenge/response-based authentication because of the entropy of the fingerprints it yields as well as the speed at which they are formed. While canvas fingerprints are generated in less than a few seconds, the link-based method proposed by [20] requires at least twenty or more seconds of data collection to be able to extract the appropriate statistical features and subsequently make an accurate classification. As a result, link-based fingerprinting may not be suitable to be used in a challenge/response-based authentication protocol despite it too being an actively generated device feature. However, it may be sufficient as a form of continuous identification following a more secure multi-factor authentication protocol. As such, link-based fingerprinting may have better applications, such as recognizing session hijacking attacks.

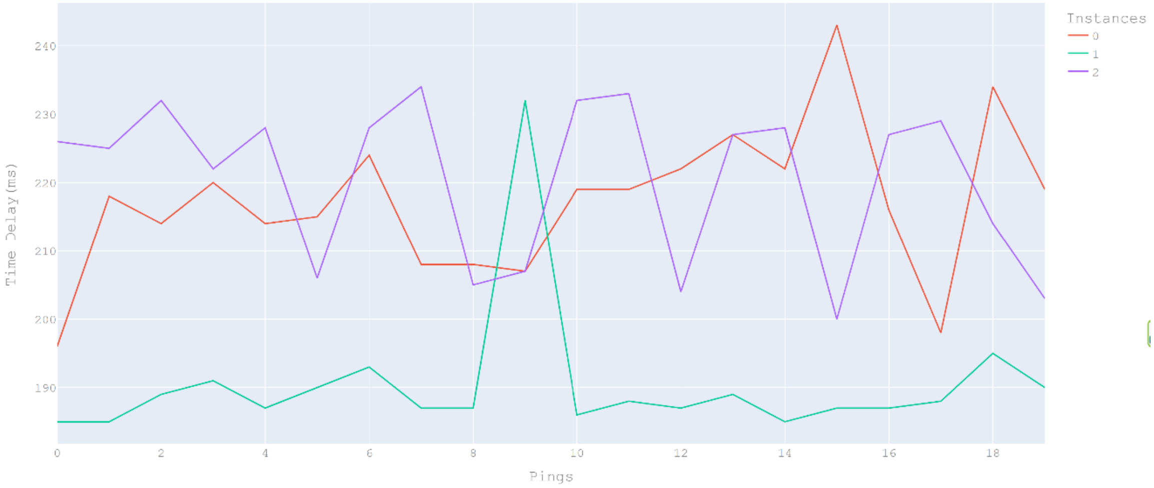


Figure 6.1: A graph of real ping traces of a few clients to one server. These are traces with high variance and are less stable than most.

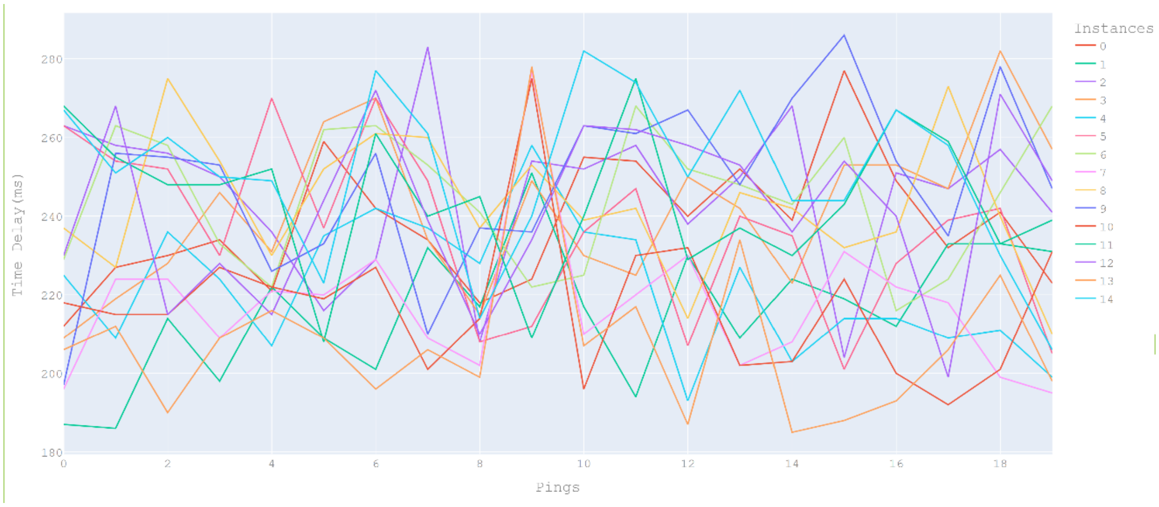


Figure 6.2: A graph of synthetic traces derived from the real traces in Figure 6.1. The synthetic traces do not appear to reflect the statistical features of the real traces.

Chapter 7: Future Works

In this work, we propose four additional features that we theorize may improve the accuracy of the PingLoc prototype. However, we were unable to reliably test the impact these features have on model training. Future works should include experiments to consider the effectiveness of these features.

Another interesting application for link-based fingerprinting could be in VPN detection. A VPN tunnel may increase the collected time delays depending on geographical distance. In theory, a user in New York connected to servers in Los Angeles would likely experience higher pings than the local Los Angeles users. Thus, it may be possible to train a machine-learning model to recognize this overhead with link-based fingerprinting.

Finally, to our knowledge, only two works discuss the formalization of browser fingerprinting [9, 16]. In [9], Lanze et al. propose a formal model capable of representing any type of fingerprinting technique from biometrics to various digital applications. Their formal model provided defined terminology allowing different fingerprinting methods to be compared more easily. The flexibility of their formal model is vital to comparing fingerprinting methods that operate in fundamentally different ways. Although the formal definitions by Pugliese et al. [16] are less extensive and less flexible, they provide a crucial insight; the evolution of fingerprints over time. As the latter does not reference the former, we believe a synthesis of the two formal models may be beneficial.

REFERENCES

[1] Turky N Alotaiby et al. "ECG-based subject identification using statistical features and random forest". In: *Journal of Sensors* 2019 (2019), pp. 1–13.

[2] Amit Datta, Jianan Lu, and Michael Carl Tschantz. "Evaluating anti-fingerprinting privacy enhancing technologies". In: *The World Wide Web Conference*. 2019, pp. 351–362.

[3] MDN Web Docs. *Cross-Origin Resource Sharing (CORS)*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (vis- ited on 03/02/2023).

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

[5] Imane Fouad et al. "Did I delete my cookies? Cookies respawn- ing with browser fingerprinting". In: *CoRR* abs/2105.04381 (2021). arXiv: 2105.04381. URL: https://arxiv.org/abs/2105.04381.

[6] Umar Iqbal et al. "Adgraph: A graph-based approach to ad and tracker blocking". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 763–776.

[7] Soroush Karami et al. "Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting." In: *In Proceedings of the 27th Network and Distributed System Security Symposium (NDSS)*. 2020.

[8] Amin Faiz Khademi, Mohammad Zulkernine, and Komminist Weldemariam. "An empirical evaluation of web-based fingerprinting". In: *Ieee Software* 32.4 (2015), pp. 46–52.

[9] Fabian Lanze, Andriy Panchenko, and Thomas Engel. "A Formalization of Fingerprinting Techniques". In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. IEEE. 2015, pp. 818–825.

[10] Pierre Laperdrix et al. "Browser Fingerprinting: A Survey". In: *ACM Trans. Web* 14.2 (Apr. 2020). ISSN: 1559-1131. DOI: 10 . 1145 / 3386040. URL: https://doi.org/10.1145/3386040.

[11] Pierre Laperdrix et al. "Fingerprinting in Style: Detecting Browser Extensions via Injected Style Sheets." In: *USENIX Security Symposium*. 2021, pp. 2507–2524.

[12] Pierre Laperdrix et al. "Morellian analysis for browsers: Making web authentication stronger with canvas fingerprinting". In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2019, pp. 43–66.

[13] Song Li and Yinzhi Cao. “Who touched my browser fingerprint? A large-scale measurement study and classification of fingerprint dynamics”. In: *Proceedings of the ACM Internet Measurement Conference*. 2020, pp. 370–385.

[14]X. Lin et al. “Fashion Faux Pas: Implicit Stylistic Fingerprints for Bypassing Browsers’ Anti-Fingerprinting Defenses”. In: *2023 2023 IEEE Symposium on Security and Privacy (SP) (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 1640–1657. DOI: 10 . 1109 / SP46215 . 2023 . 00094. URL: https : / / doi . ieeecomputersociety.org/10.1109/SP46215.2023.00094.

[15] Phu H. Phung et al. “A User-Oriented Approach and Tool for Security and Privacy Protection on the Web”. In: (June 2020). DOI: 10.1007/ s42979-020-00237-5.

[16] Gaston Pugliese et al. “Long-Term Observation on Browser Fingerprinting: Users’ Trackability and Perspective.” In: *Proc. Priv. Enhancing Technol.* 2020.2 (2020), pp. 558–577.

[17] Jordan S Queiroz and Eduardo L Feitosa. “A web browser fingerprinting method based on the web audio API”. In: *The Computer Journal* 62.8 (2019), pp. 1106–1120.

[18] Konstantinos Solomos et al. “The dangers of human touch: fingerprinting browser extensions through user actions”. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 717–733.

[19] Antoine Vastel et al. “Fp-stalker: Tracking browser fingerprint evolutions”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 728–741.

[20] Tianqi Wu et al. “My site knows where you are: a novel browser fingerprint to track user position”. In: *ICC 2021-IEEE International Conference on Communications*. IEEE. 2021, pp. 1–6.

[21] Desheng Zhang et al. “A Survey of Browser Fingerprint Research and Application”. In: *Wireless Communications and Mobile Computing* 2022 (2022).