Universidade Federal do Amazonas
Instituto de Computação
Programa de Pós-Graduação em Informática

Jordan de Sá Queiroz

Um Método de *Web Fingerprinting* baseado em Atributos de Hardware

Manaus
Abril de 2018

Jordan de Sá Queiroz

Um Método de *Web Fingerprinting* baseado em Atributos de Hardware

Trabalho apresentado ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas como requisito para obtenção do grau de mestre.

Orientador: Prof. Dr. Eduardo Luzeiro Feitosa

Manaus
Abril de 2018

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Queiroz, Jordan de Sá
Q3u Um Método de Web Fingerprinting baseado em Atributos relacionados ao Hardware / Jordan de Sá Queiroz. 2018
84 f.: il. color; 31 cm.

Orientador: Eduardo Luzeiro Feitosa
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Web fingerprinting. 2. HTML5 Canvas. 3. Web Audio API. 4. Privacidade. I. Feitosa, Eduardo Luzeiro II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Um Método de Web Fingerprinting baseado em Atributos relacionados ao Hardware"

JORDAN DE SÁ QUEIROZ

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Eduardo Luzeiro Feitosa - PRESIDENTE

Prof. Edleno Silva de Moura - MEMBRO INTERNO

Prof. Rafael Roque Aschoff - MEMBRO EXTERNO

Manaus, 23 de Março de 2018

Epígrafe

It is your new best friend, new eye in your private den
And you can find the world inside it every day
They know what you did today, hear everything you say
And when they send you mail, they know where you will go sometime next day
They know your life, they have a file about you.

Blank File – Sonata Arctica

Agradecimentos

Agradeço a Deus pela oportunidade e capacidade de poder cursar o mestrado. Agradeço também ao meu orientador, Prof. Dr. Eduardo Luzeiro Feitosa, que me auxiliou de forma muito paciente e acreditou no potencial desta pesquisa.

Agradeço a minha família pelas condições que me deram de poder estudar, sem eu precisar me preocupar em trabalhar. Sem essa ajuda teria sido muito mais difícil. Também agradeço a minha namorada, Priscila Barros que, assim como a minha família, compreendeu os momentos em que estive ausente.

Agradeço também aos amigos que me ajudaram em diversas etapas do mestrado. Ádria Meneses, Adriana Saraiva, Pablo Elleres, Rayol Neto, Hendrio Luis, Quevin Quispe, Klinsman Maia, Edma Urtiga e outras pessoas que ajudaram direta ou indiretamente, principalmente durante os experimentos em que eram necessários voluntários.

Aos professores e ao corpo administrativo do PPGI pelo suporte e acompanhamento durante esta fase da minha formação acadêmica.

Por último, mas não menos importante, agradeço a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) por também dar suporte a pesquisa desenvolvida neste trabalho.

Resumo

Web Fingerprinting é o processo no qual um usuário é, com alta probabilidade, identificado de forma única a partir das características extraídas de seu dispositivo, gerando uma chave identificadora (*fingerprint*). Para um método que gere um *fingerprint* ser eficaz é necessário obter respostas estáveis, o que implica em empregar atributos discriminatórios com baixa volatilidade. Em outras palavras, atributos capazes de fornecer as mesmas características sobre os dispositivos ao longo do tempo. Há uma diversidade de técnicas propostas na literatura, mas nem todas são capazes de gerar um *fingerprint* estável. Nesta dissertação é proposto, projetado e avaliado um método de *Web Fingerprinting* que busca utilizar características relacionadas ao hardware dos dispositivos. Uma das formas de alcançar esse objetivo é empregar HTML5 Canvas e Web Audio API, tecnologias promissoras por serem capaz de fornecer características relacionadas ao hardware do dispositivo, o que reduz a mutabilidade do *fingerprint* extraído e aumenta o número de dispositivos-alvo em que o método pode ser aplicado. Como resultado, constatou-se que o emprego do HTML5 Canvas e da *Web Audio API*, em conjunto como outros atributos cujas características são relativas ao hardware do dispositivo, permite identificar, de forma única, com 90,34% de precisão, diversos usuários. Além disso, percebeu-se que agrupamento de atributos mais fracos com os mais discriminatórios permite extrair mais características do que utilizar atributos discriminatórios de forma isolada.

Palavras-chave: Web fingerprinting, HTML5 Canvas, Web Audio API.

Abstract

Web fingerprinting is the process in which a user is, with high likelihood, uniquely identified by the extracted features from his/her device, generating a fingerprint. In order to be effective, the method must generate a stable fingerprint, and therefore it is necessary to employ discriminatory attributes with low volatility, capable of providing the same characteristics over the time. There are a variety of proposed techniques, but not all of them are capable of generating a stable fingerprint. In this work it is proposed, designed and evaluated a Web Fingerprinting method that aims to employ features that provide characteristics related to the devices' hardware. One of the ways to achieve this objective is through the use of technologies such as HTML5 and the Web Audio API. Theses are promising technologies for Web Fingerprinting methods because they provide features related to the devices' hardware, which reduces the extracted fingerprint's mutability and increases the number of target devices in which the method can be executed, since HTML5 is adopted by default in the most popular web browsers. As results, it was found that the HTML5 Canvas and the Web Audio API, when employed with other attributes related to the hardware characteristics of the device, converges to a web Fingerprinting method capable of uniquely identify several users (with 90,34% of accuracy). In addition, it was found that grouping weaker attributes with more relevant ones allows the Web Fingerprinting method to extract more characteristics than to use just relevant attributes.

Keywords: Web fingerprinting, HTML5 Canvas, Web Audio API.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Atualmente, a *Web* é o meio mais utilizado para a realização de atividades como transações bancárias, comunicação em mídias sociais, compartilhamento de imagens e vídeos, dentre outras [1]. Todo esse conjunto de possibilidades foi alcançado devido ao rápido avanço das tecnologias *Web* (linguagens, APIs, dentre outras), que proporcionam experiências de uso cada vez melhores. Ironicamente, tais avanços prejudicam a privacidade na *Web*, pois permitem que os usuários possam ser rastreados e identificados por atacantes e companhias de anúncio de digital, por exemplo [2]. Uma das recentes formas de realizar essa identificação se baseia no processo de coletar **características discriminatórias** presentes nos dispositivos dos usuários, podendo, assim, identificar unicamente um dispositivo e, consequentemente, seu dono. Esse processo é denominado como *Web Fingerprinting* e pressupõe que cada usuário possui seu próprio dispositivo de uso pessoal.

Assim como os *cookies*[1], os métodos de *Web Fingerprinting* também são usados para rastrear usuários. Contudo, diferentemente dos *cookies*, podem ser usados para protege-los, atuando, por exemplo, com um fator a mais na autenticação em múltiplos fatores de usuários [16], evitando ataques a lojas de aplicativos [5] e remediando roubo de sessão. Por outro lado, os métodos de *Web Fingerprinting* tem todo potencial para serem empregados com o objetivo de ferir a privacidade dos usuários sem a necessidade manter estados no dispositivo do cliente [3, 4].

Embora a literatura acadêmica apresente dezenas de métodos de *Web Fingerprinting* (por exemplo, [2, 5, 6, 7, 8, 9, 10, 11, 12, 13]), eles empregam características com alto grau de mutabilidade ou que dependem de informações providas pelo navegador e seus *plugins* e/ou extensões. Em outras palavras, eles usam características que podem mudar por diferentes fatores, como novas instalações e

[1] Arquivos cujo conteúdo são pequenas cadeias de caracteres usadas para registrar dados relacionados a sessões *Web*.

atualizações de software, por exemplo. Tais mudanças resultam na instabilidade do *fingerprint* obtido, dificultando o processo de (re)identificar um usuário [2, 7].

Então, quais características usar para se obter um *fingerprint* correto? Nesta dissertação é proposto, projetado e avaliado um novo método de *Web Fingerprinting* que permite a obtenção de características relacionadas ao hardware do dispositivo do usuário, reduzindo a mutabilidade do *fingerprint* extraído e aumentando o número de dispositivos-alvo em que o método pode ser aplicado.

1.1 Motivação

Navegar na *Web* tornou-se uma atividade cotidiana na vida do ser humano. No entanto, na medida em que os usuários o fazem, seus perfis e informações privadas podem ser coletados ou inferidos por meio do registro de suas atividades *online*. O primeiro meio encontrado para rastrear os hábitos de navegação dos usuários foram os *cookies* [3].

Cookies são instanciados pelos sites e armazenados no dispositivo do usuário. Tipicamente, seus conteúdos são referentes aos dados de sessões *Web* (por exemplo, dados de *login*). Como podem servir para rastrear as atividades dos usuários na *Web*, os *cookies* acabaram por incentivar a comercialização de dados dos usuários entre empresas de *marketing* digital. Uma vez que os dados dos *cookies* eram usados sem qualquer critério, ferindo a privacidade dos usuários, leis foram criadas para definir e estabelecer como os sites devem utilizá-los, bem como os dados contidos [14]. Tais medidas contribuíram para que as companhias interessadas em rastrear usuários buscassem novas formas de rastreamento [3]. A solução encontrada foi utilizar técnicas de *Web Fingerprinting*, pois são invisíveis para o usuário, não possuem a necessidade de armazenar dados no dispositivo [15] e nem possuem qualquer tipo de legislação restritiva.

A questão envolvendo as técnicas de *Web Fingerprinting* é a estabilidade. Tomando como exemplo o uso na autenticação em múltiplos fatores, um método que depende de softwares atualizados diariamente, por exemplo, impossibilitaria a autenticação do usuários, visto que seu *fingerprint* mudaria constantemente. De modo contrário, o que se espera é a estabilidade do *fingerprint* a longo prazo. Desta forma, percebe-se que métodos estáveis de *Web Fingerprinting* são capazes de contribuir não a segurança como com a comodidade do usuário. É com base na questão da estabilidade que a solução proposta nesta dissertação investiga uma área em que existem poucas pesquisas relevantes em andamento.

Além disso, para alcançar melhores resultados, existem questões de pesquisa que impulsionam este trabalho:

- Quais são os atributos ou informações que estão estritamente relacionados às características de hardware?

- Quais são as técnicas propostas na literatura de *Web Fingerprinting* que utilizam características relacionadas ao hardware?
- É realmente possível identificar, de forma única, os usuários apenas empregando características fortemente relacionadas ao hardware do dispositivo?

1.2 Hipótese

O uso de características que estão relacionados ao hardware, por exemplo, Canvas, Web Audio API e WebGL, reduz a dependência por características de software. Dessa maneira, é possível implementar um método de *Web Fingerprinting* que explora tais características, a fim de obter um identificador com alta probabilidade de ser único e estável ao longo do tempo.

1.3 Objetivos

Projetar e avaliar um método para realizar *Web Fingerprinting*, empregando e combinando propriedades e atributos fortemente relacionados ao hardware do dispositivo, a fim de identificar de forma única os usuários. Para tanto, elementos já consolidados do HTML5 e relacionados ao dispositivo (*Web Audio API*, WebGL, Canvas, dentre outros) serão utilizados, visando a aplicação em cenários reais de forma a medir sua efetividade. No quesito avaliação, o conceito de entropia será usado como métrica principal de validação.

Especificamente, pretende-se:

- Comparar os métodos de Canvas *fingerprinting* propostos na literatura e selecionar o que mais se adequa à solução proposta neste trabalho, por meio de experimentos empíricos;
- Desenvolver um algoritmo para gerar um identificador único para o dispositivo do usuário, a partir de suas características extraídas.

1.4 Contribuições esperadas

Como contribuição, espera-se:

- Demonstrar, por meio de experimentos empíricos, que um artefato de *Web Fingerprinting* baseado em atributos fortemente relacionados ao hardware é tão eficaz (ou mais) quanto aqueles que apenas se valem de características relacionadas a software.

1.5 Estrutura do Documento

Este documento está organizado em 8 capítulos. No capítulo 2 são apresentados os conceitos básicos necessários para compreensão desta proposta. O capítulo 3 expõe os trabalhos relacionados encontrados na literatura, dando um foco na questão de características extraídas a partir do HTML5 Canvas e do *Web Audio API*. O Capítulo 4 apresenta a avaliação da *Web Audio API* e do Canvas. O capítulo 5 detalha a proposta e sua metodologia, apresentando uma arquitetura, bem como os atributos inicialmente selecionados para realizar o *fingerprinting* em dispositivos. O Capítulo 6 apresenta o experimento realizado para avaliar o método e os resultados obtidos. O capítulo 7 apresenta como o método reage às contramedidas ao *Web Fingerprinting*. E, por fim, o Capítulo 8 conclui este trabalho.

Capítulo 2

Conceitos básicos

Este Capítulo apresenta os principais conceitos básicos necessários para a compreensão dos temas abordados nesta dissertação como *Web Fingerprinting*, HTML5 Canvas e *Web Audio API*.

2.1 *Web Fingerprinting*

Web fingerprinting é o processo no qual as características de um dispositivo são extraídas para gerar uma chave identificadora (*fingerprint*), capaz de identificar, com alta probabilidade e de forma única, um usuário. A Figura 2.1 ilustra e explana a dinâmica de um *Web fingerprinting*.

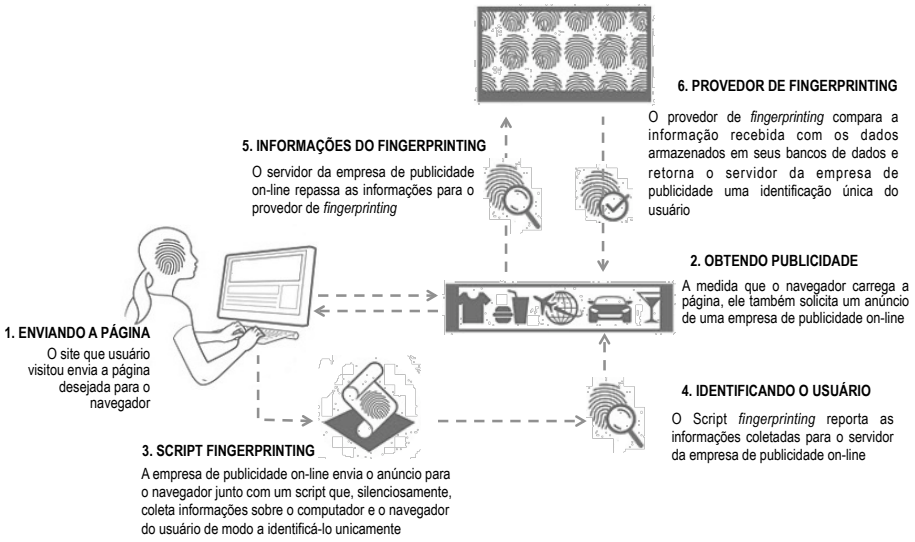


Figura 2.1: Dinâmica de um *Web fingerprinting* [17].

No exemplo da Figura 2.1, o usuário visita um site qualquer (1), enquanto o site é renderizado no navegador, as propagandas online também são carregadas (2). Depois que a página está carregada, o *script* de *Web Fingerprinting* é executado, obtendo as características do dispositivo (3). Após a execução do *script*, as características coletadas são enviadas para o servidor da empresa de publicidade online (4), que por sua vez repassa os dados para a provedora do *Web Fingerprinting* (5). Por fim, a provedora do *Web Fingerprinting* processa os dados recebidos, salvando-os em um banco de dados e retornando uma identificação única do usuário para a empresa de publicidade (6).

De acordo com a W3C (W*orld Wide Web Consortium*) [18], *Web Fingerprinting* pode ser classificado em três tipos: passivo, ativo e *cookie-like*. O **passivo** é baseado nas características observáveis das solicitações *Web* (cabeçalhos HTTP, endereço IP e outras informações do nível de rede), sem utilização de qualquer código executando no lado do cliente. O **ativo** utiliza códigos dinâmicos (JavaScript, Action Script, dentre outros) no lado do cliente para observar características sobre o navegador. Dentre aquelas observáveis podem-se destacar a lista de fontes disponíveis no navegador, o tamanho da tela, lista de *plugins*, padrões na renderização de gráficos, dentre outros. O terceiro e último tipo é o ***cookie-like***, onde um código ou aplicativo é instalado no dispositivo do usuário, permitindo sua (re)identificação e inferências sobre o mesmo [17].

2.1.1 Características de Software e Hardware

As características que podem ser obtidas do dispositivo podem ser classificadas em relacionadas ao software (navegador *Web*) e ao hardware (periféricos do dispositivo), podendo ser extraídas a partir dos métodos de uma linguagem de programação (JavaScript na quase totalidade dos trabalhos). As características relacionadas ao software são aquelas que dependem das configurações do navegador *Web*, seus complementos (extensões, *plugins*, tipos MIME e fontes instaladas, por exemplo) e de seu ambiente de execução, como, por exemplo, sistema operacional, versão, nome e plataforma do navegador, dentre outras. O problema dessas características é que elas podem ser alteradas por diversos fatores: atualização do navegador, instalação de novos *plugins*, novas fontes e até o uso de extensões de contramedida.

Já as características relacionadas ao hardware são aquelas que dependem dos periféricos e até mesmo dos *drivers* instalados no dispositivo em que o navegador *Web* está em execução. Dessa forma, a alteração de tais características ocorrem com rara frequência. Dentre alguns exemplos, pode-se destacar: monitor/tela do dispositivo[1], placa gráfica, *drivers* e placa de áudio [1, 7, 19].

[1]Embora a dimensão de tela seja uma característica relacionada ao software, o método

A *Web* contém diversas tecnologias que podem ser empregadas em técnicas para realizar *Web fingerprinting*, existindo um amplo leque delas já consolidadas: JavaScript, Flash, HTML5 Canvas, *cookies*, Cabeçalhos HTTP, CSS e HTML. Outras ainda em estágio de desenvolvimento[2] e, consequentemente, sem um padrão adotado: *Web Audio API*. Alguns meios disponíveis para obter características são: *userAgent*, *javaEnabled*, *width*, *toDataURL*, *maxTouchPoints*, dentre outros [1, 4, 12, 15].

2.1.2 Tecnologias Tradicionais Aplicadas ao *Web Fingerprinting*

As tecnologias mais adotadas na literatura de *Web fingerprinting* são:

- **JavaScript**: Permite acesso fácil e direto a diferentes APIs que fornecem informações relevantes sobre o navegador e dispositivo em que o primeiro está sendo executado (por exemplo, lista de *plugins* e sistema operacional) [6, 20].
- **Flash**: Assim como JavaScript, provê métodos que possibilitam o acesso a diversas informações que estão estritamente relacionadas ao sistema em que o navegador está sendo executado. Por exemplo, podem ser obtidos o nome do sistema operacional, arquitetura da CPU, número de DPI (*Dots Per Inch*) da tela, dentre outras características [20].
- **CSS**: Diferentes versões e implementações do CSS nos navegadores *Web* podem ser utilizadas para avaliar diferenças no motor de *layout* dos mesmos, a fim de identifica-los. [12].
- **Cabeçalho HTTP**: Existem uma série de informações que caracterizam um navegador e são suficientes para sua identificação devido a flexibilidade permitida pela RFC 2616, que admite distinções no número, conteúdo e ordem dos campos do cabeçalho HTTP utilizados pelos navegadores nas requisições. Alguns dos campos que podem ser extraídos do cabeçalho HTTP são: *UserAget*, *Accept*, *Accept-Language*, *Accept-Encoding*, endereço IP, dentre outros cabeçalhos especificados pela RFC 2616 [12].

proposto neste trabalho não a utiliza devido sua volatilidade quando faz-se uso de monitores secundários. Além disso, a dimensão pode ser forjada com o uso de extensões instaladas no navegador *Web*.

[2] Até a data de escrita deste trabalho.

2.1.3 Tecnologias de *Web Fingerprinting* relacionadas à Software e Hardware

Dentre as tecnologias atuais que permitem a geração de *fingerprints* e possuem relação tanto com o software quanto com o hardware do dispositivo, pode-se destacar duas: Canvas e Web Audio API.

Canvas *fingerprinting*

O HTML5 Canvas[3] é um elemento usado para criar desenhos dinâmicos com o uso do JavaScript, provendo diversos métodos de desenho: caixas, círculos, textos e inclusão de imagens [21]. O desenho é formado por *pixels*, sendo que a organização destes dependem do motor de renderização do navegador *Web*, sistema operacional, placa gráfica do dispositivo e o *driver* de vídeo instalado na mesma. Após a renderização do desenho, a representação do mesmo pode ser extraída a partir das seguintes funções: *getImageData()* e *toDataURL()*. A primeira provê os valores RGBA[4] (em inteiro) para cada *pixel* de uma área do Canvas. A segunda função retorna a imagem renderizada na forma de uma URL codificada em Base64, onde estão contidos os dados (*pixels*) da mesma [9]. Esses dados contribuem para identificar um dispositivo de forma única e, consequentemente, seu dono. O quanto mais complexa for a imagem, mais chances de ser único será o *fingerprint* obtido [9].

A Figura 2.2 ilustra a dinâmica de funcionamento do Canvas *fingerprinting* no contexto 2D. No trecho de código 2.1 está um exemplo de Canvas *fingerprinting*.

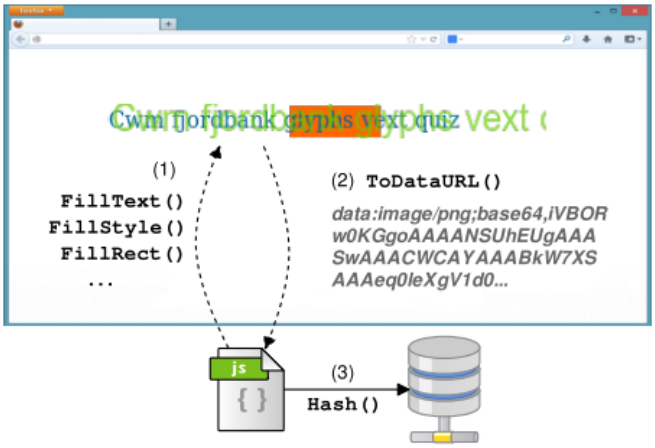


Figura 2.2: Dinâmica de funcionamento do HTML5 Canvas *fingerprinting* [22].

[3] http://www.w3schools.com/html/html5_canvas.asp.
[4] https://www.w3schools.com/cssref/css_colors_legal.asp.

```
1  var canvas = document.getElementById("drawing");
2  var context = canvas.getContext("2d");
3  context.font = "18pt Arial";
4  context.textBaseLine = "top";
5  context.fillText("Hello, user", 2, 2);
6  var encondedImage = canvas.toDataURL();
```

Listing 2.1: Exemplo do Canvas Fingerprinting no contexto 2D [9]

Como ilustrado na Figura 2.2, para utilizar o Canvas como um método de Web Fingerprinting, é necessário utilizar funções de desenho, como exemplificado no passo 1. Após a imagem ser renderizada, a mesma pode ser obtida com uma função que extrai a representação da imagem num formato de URL, como está ilustrado no passo 2. Depois que a representação é obtida, a mesma é aplicada em uma função de *hashing*, cujo resultado é usado como *fingerprint*.

Embora pareça um recurso inofensivo, o Canvas pode ser usado em métodos de *Web Fingerprinting*, devido as imagens dependerem de como a GPU organiza os pixels. Dessa maneira, o *fingerprint* obtido depende de como a GPU trabalha e, por tanto, está relacionado com características de *hardware*.

Web Audio API

O *AudioContext* é uma das interfaces da *Web Audio* API que representa um grafo de processamento de áudio, construído a partir de vários nós interligados. Essa interface controla a criação dos nós de áudio contidas em si, a execução do processamento de áudio e/ou a decodificação do mesmo. A Figura 2.3 ilustra em alto nível a dinâmica de funcionamento do *AudioContext*.

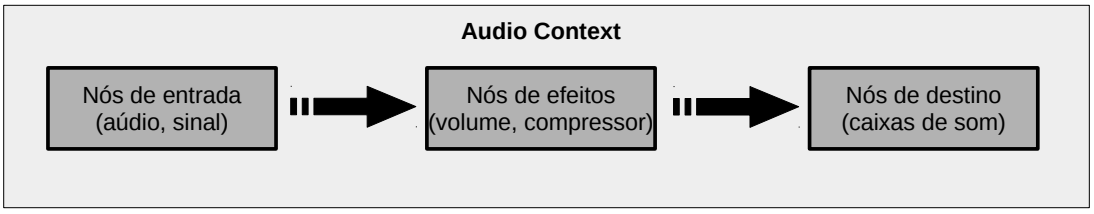


Figura 2.3: Dinâmica de funcionamento do *AudioContext* [23]. Após essa sequência o áudio deve ser processado e o *fingerprint* obtido [15].

Existem várias funções, APIs e interfaces que podem ser utilizadas para projetar um método de *Web Fingerprinting* com AudioContext, todas baseadas na documentação apresentada por [23].

- **AudioNode**: Interface que representa um módulo de processamento de áudio, por exemplo, uma fonte/destino de áudio ou módulo de processa-

mento intermediário. O método *connect()* encontrado neste nó, permite conectar a saída de um nó *A* ao *B*.

- **OscillatorNode**: Interface que representa um módulo de processamento de áudio que gera uma onda periódica (sinal) com determinada frequência em *hertz*. As ondas podem ser dos seguintes tipos: Senoidal, quadrada, triangular e customizada.

- **GainNode**: Interface que representa um módulo de processamento de áudio que possibilita a alteração do volume de uma fonte sonora de entrada antes que a mesma se propague para outros nós.

- **AudioDestinationNode**: Interface que representa o destino final em um grafo de áudio instanciado num dado contexto, geralmente o destino é a caixa de som do dispositivo.

- **AnalyserNode**: Interface que representa um nó capaz de prover, em tempo real, informações de análise no domínio da frequência e do tempo, para o propósito de visualização dos dados de áudio. Este nó não necessita estar conectado à um nó de saída para prover os dados. Essa interface possui dois métodos que provêm dados sobre o áudio (I) *getByteFrequencyData(Uint8Array)*: Copia os dados de frequência do som (ou sinal) para vetor passado por parâmetro; (II) *getByteTimeDomainData(Uint8Array)*: Copia os dados de frequência do sinal (no domínio do tempo) para o vetor passado por parâmetro.

- **ScriptProcessorNode**: É a interface que representa um módulo de processamento de áudio, permitindo gerar, processar ou analisar áudio com o JavaScript. Esse módulo está interligado a dois *buffers*, um contendo dados de entrada e o outro cujo conteúdo são os dados de saída. Esse módulo possui um manipulador de eventos (*onaudioprocess()*), que é disparado sempre que o *buffer* de entrada está pronto para ser processado.

- **AudioBuffer**: É a interface que representa um pequeno recurso de áudio carregado em memória, obtido a partir de um áudio decodificado ou de dados (frequências) brutos.

- **ChannelMergerNode**: É uma interface que recebe como entrada diversos sinais em diferentes canais, juntando-os e propagando-os na saída como um único sinal.

Para utilizar qualquer um desses nós, é necessário instanciar um contexto de áudio. Neste caso, tem-se duas opções: *AudioContext* e *OfflineAudioContext*. O

AudioContext é um contexto de áudio que controla a criação de nós e a execução do processamento de áudio. O *OfflineAudioContext* tem função similar ao *AudioContext*, mas ao invés de renderizar o som diretamente no dispositivo do usuário, gera o som ou o decodifica em um *AudioBuffer*. Na prática, para escutar algum som produzido a partir de um nó oriundo (instanciado) do *OfflineAudioContext*, é necessário renderiza-lo por meio do *AudioContext*. No caso de nós oriundos do *AudioContext*, tal procedimento não é necessário. A Figura 2.4 ilustra um exemplo simples do funcionamento do *AudioContext* (à esquerda) e do *OfflineAudioContext* (à direita) e o trecho de código 2.2 demonstra um exemplo de implementação em alto nível.

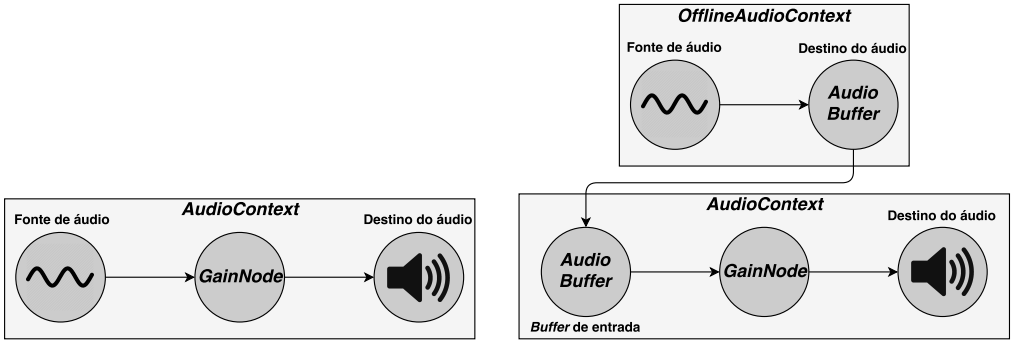


Figura 2.4: Exemplos de implementação do *AudioContext* (à esquerda) e do *OfflineAudioContext* (à direita).

```
1  // Para melhor compatibilidade com o Chrome e Safari.
2  var audioCtx = new (window.AudioContext || window.
       webkitAudioContext)();
3  // Fonte de audio.
4  var oscillator = audioCtx.createOscillator();
5  // Efeito (para controlar o volume do som).
6  var gainNode = audioCtx.createGain();
7  // O analyser permite obter as frequencias do som.
8  var analyser = audioCtx.createAnalyser();
9  // Usado para processa a onda sonora durante a execucao da
       mesma.
10 var processorNode = audioCtx.createScriptProcessor();
11 // Contectando os nos.
12 oscillator.connect(analyser);
13 analyser.connect(processoNode);
14 processorNode.connect(gainNode);
15 gainNode.connect(audioCtx.destination);
16 // Configurando o volume para 0
17 gainNode.gain.value = 0;
18 processorNode.onaudioprocess = function(audioProcessingEvent) {
```

```
19      // Obtem as frequesncias da fonte de som.
20      analyser.getByteTimeDomainData(dataArray);
21    }
22    // Iniciando o som
23    oscillator.start();
```

Listing 2.2: Exemplo em alto nível da Web Audio API com o AudioContext Fingerprinting [15, 23]

Uma aplicação que emprega o *AudioContext* funciona da seguinte maneira: (I) Um contexto de áudio (*AudioContext*) é criado; (II) Dentro do contexto, fontes de áudio são criadas; (III) Os efeitos sonoros precisam ser criados; (IV) O destino final do áudio (por exemplo, a caixa de som do dispositivo) precisa ser escolhido; (V) As fontes de áudio precisam ser conectadas com seus efeitos e, seus efeitos, conectados com o destino [23]. Esse fluxo pode ser empregado em algum método de *Web Fingerprinting*, visto que o som pode ser configurado para não ser audível no destino, ser processado e ter suas características extraídas e modeladas, resultando em um identificador único.

Por depender da placa de som, o *AudioContext* está relacionado com as características de hardware do dispositivo. De forma análoga ao Canvas *fingerprinting*, o processamento de áudio em diferentes máquinas e navegadores pode ser usado para obter características dos dispositivos devido a diversas combinações de hardware (placa de som) e software (*driver*). A ideia é um método de *Web Fingerprinting* combine processamento de áudio (hardware e software) que, mesmo quando aplicado em máquinas similares, resulte em *fingerprints* diferentes. Esse fenômeno pode ser atribuído a inconsistências durante o processo de fabricação do hardware, mesmo que seus modelos e número de versão sejam iguais. Essas inconsistências podem revelar, por exemplo, diferentes latências nas renderizações (no caso da GPU) [11].

2.2 Entropia

Entropia é a medida da incerteza do valor de uma variável aleatória. Suponha que X seja uma variável aleatória discreta com alfabeto χ e função de probabilidade $p(x) = Pr\{X = x\}, x \in \chi$. $p(x)$ se refere a probabilidade da ocorrência de x [24]. A entropia $H(X)$ de uma variável aleatória discreta X é definida pela Equação 2.1.

$$H(X) = -\sum_{x \in \chi} p(x) \log_2 p(x). \tag{2.1}$$

Trazendo essa equação para a contexto da proposta apresentada neste trabalho, $H(X)$ é a entropia da amostra de *fingerprints*, x é um fingerprint contido em

χ, que por sua vez é o conjunto de chaves identificadoras. O quanto maior for a entropia da amostra, mais distintos serão os *fingerprints* e, consequentemente, maiores as chances de identificar unicamente um usuário. Vale ressaltar que o logaritmo é na base 2 e a entropia é expressada em *bits* [24].

Métodos de *Web Fingerprinting* geram chaves identificadoras com base nas características dos dispositivos, ou seja, dependendo daquelas obtidas, uma população de dispositivos contribuirá para uma amostra de diversas chaves distintas. O grau de distinção entre as chaves ou informação obtida por meio das mesmas, como apontado por Khademi et al. (2015), pode ser mensurado com o uso do conceito de entropia.

Devido as propostas de *Web Fingerprinting* relacionadas com este trabalho empregarem o conceito de entropia apresentado na Equação 2.1 e tendo em vista a capacidade da mesma de medir o grau da distinção entre elementos, o conceito de entropia é adotado como mecanismo de avaliação e comparação.

2.3 Considerações do Capítulo

Atributos que dependem do hardware, como o Canvas e *AudioContext* (este último presente na *Web Audio API*), podem ser potencialmente explorados por diversas entidades interessadas em identificar usuários. Pelo fato do HTML5 ser adotado pelos navegadores mais populares, um método de *Web Fingerprinting* que faz uso de tais recursos possui mais chances de executar com sucesso em diversos dispositivos com capacidade de navegação. Dessa forma, espera-se que a extração de característica dos dispositivos, por meio de atributos que provêm características de hardware, seja uma maneira eficaz de projetar um artefato de *Web Fingerprinting* capaz de identificar, de forma única, usuários na *Web*.

Capítulo 3

Trabalhos Relacionados

A literatura acadêmica compreende diversos trabalhos sobre *Web Fingerprinting*, dentre os quais existem várias propostas de métodos para extrair o *fingerprint* de dispositivos. As abordagens se valem do uso de atributos relacionados a características de software e hardware. Este Capítulo apresenta os trabalhos relacionados que empregam o HTML5 Canvas e o *AudioContext* em seus métodos, mesmo que neles estejam presentes propriedades correlatas a software. Os trabalhos estão organizados por seções e ordenados cronologicamente.

3.1 Pixel Perfect

Mowery e Shacham (2012) [9] propuseram um método de *Web Fingerprinting* baseado na renderização de texto e gráfico em 3D no HTML5 Canvas. A ideia foi que como cada navegador possui um conjunto de fontes (muitas vezes customizadas e adicionadas pelo usuário), a combinação do HTML5 Canvas com fontes é capaz de extrair características discriminatórias que podem ser usadas para implementar um *Web Fingerprinting* [9]. Para tanto, os autores renderizam um pantograma[1] no HTML5 Canvas. Embora a imagem resultante seja a mesma em diversos navegadores, existem diferenças na organização de seus *pixels*, sendo elas não notáveis a olho nu [9]. A Figura 3.1 ilustra diversas organizações dos *pixels* quando a mesma imagem é renderizada em três versões distintas do Windows: XP, Vista e 7.

Embora seja possível extrair características relevantes dos dispositivos com desenhos 2D , os autores também fizeram uso do HTML5 Canvas para renderizar gráficos em 3D, neste caso, usando WebGL. O WebGL é um padrão multiplataforma para o desenho de gráfico em 3D na *Web* baseado no OpenGL ES, sem a

[1] Sentença que é formada por todas as letras do alfabeto de uma língua.

Figura 3.1: Mapa de pixels durante a renderização de um pantograma no Windows XP, Vista e 7, respectivamente [9].

necessidade de utilizar *plugins* para tal, implementado por padrão nos navegadores Chrome, Firefox, Safari e Opera [9, 25]. A Figura e 3.2 ilustra um exemplo de uma imagem renderizada a partir do WebGL (figura mais à esquerda) e seu mapa de *pixels* em três placas gráficas distintas.

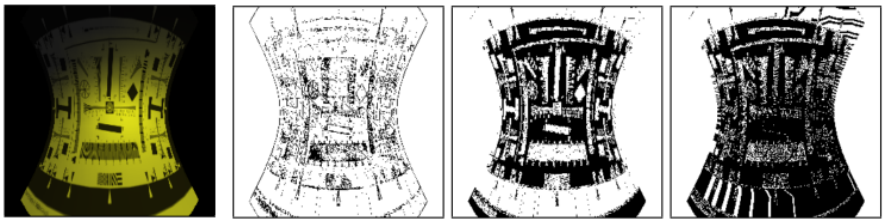


Figura 3.2: Mapas de *pixels* resultantes das renderizações da figura mais à esquerda em três placas gráficas distintas, nesta ordem: Radeon HD 2400, Intel 82945G e Intel G33/G31 [9].

Para avaliar, Mowery e Shacham empregaram entropia para medir o grau de aleatoriedade obtido a partir das características de cada dispositivo. Quanto maior o grau de entropia (em *bits*), menor é a chance de dois ou mais dispositivos compartilharem o mesmo *fingerprint*. Os autores argumentam também que quando se trata da renderização de gráficos mais complexos, o grau de entropia é maior. A entropia é calculada pela fórmula de Shannon [26].

O resultado do experimento de Mowery e Shacham (2012) [9] consistiu em 116 dispositivos identificados unicamente de um total de 294. A amostra obtida apresenta entropia de 5,73 bits para o Canvas nos contextos 2D e 3D. No caso do HTML5 Canvas exclusivamente no contexto 2D, foi alcançada a entropia de grau 3,05 bits. As renderizações foram agrupadas por grau de similaridade.

Embora a proposta cumpra o objetivo de identificar, de forma única, os usuários, no método dos autores apenas um atributo relacionado a característica de hardware foi utilizado, nesse caso, um atributo que depende da GPU. Utilizar

uma característica que está relacionada com a placa de áudio pode contribuir para identificar, de forma única, mais usuários.

3.2 Hardware Fingerprinting Using HTML5

Métodos de *hardware Fingerprinting* empregando o HTML5 são propostos e analisados por Nakibly et al. (2015) [11], dos quais somente o que usa Canvas foi implementado. Diferentemente das outras técnicas, que se valem do Canvas, os autores tiveram como objetivo obter o *fingerprint* da GPU do dispositivo com base em duas características: frequência de *clock* da CPU e viés do *clock* da GPU. O *clock* da CPU é usado como referência para medir o viés do *clock* da GPU, ou seja, o *fingerprint* é obtido a partir da medição do viés dos *clocks* da CPU e GPU.

Embora o JavaScript possua métodos que retornam o *timestamp* do *clock* da CPU, o mesmo não acontece para a GPU. Para resolver esse problema, os autores mensuram o número de *frames* renderizados na tela do dispositivo em um intervalo de tempo predefinido. Essa mensuração serviu de base para o *fingerprint* da GPU. Assumindo que o método de *fingerprinting* é o único processo que está demandando recursos da GPU, então o número de *frames* renderizados vai depender fortemente das características da mesma (frequência de *clock*, número de núcleos e outros parâmetros relacionados ao seu desempenho) [11].

Durante o experimento, as renderizações são divididas em três fases, cada uma com duração de 15 segundos, sendo que cada fase é composta por 3 animações de 5 segundos. Conforme seus avanços, a complexidade das renderizações também aumenta, exigindo mais recursos do hardware. No total, o processo consome 45 segundos [11].

Nakibly et al. (2015) [11] demonstram que, o quanto mais complexas forem as animações, mais características serão extraídas da GPU e mais eficaz será o método de *Web Fingerprinting*. No total, 130 dispositivos participaram do experimento. Além disso, os autores executaram o experimento em 9 máquinas cuja configuração de hardware e software são as mesmas e, mesmo assim, cada uma delas foi identificada unicamente. Também são reportados os graus de entropia na primeira, segunda e terceira fase do experimento, respectivamente: 2,8; 5,03 e 5,14 *bits*.

Embora o método dos autores se destaca por ser capaz de identificar, de forma única, maquinas clonadas, o método proposto por [11] exige muito do hardware ao ponto de ser perceptível que o processador e o navegador estão sob carga pesada de trabalho. Esse efeito pode denunciar ao usuário que há algo de errado na página Web e, dessa forma, levantar suspeitas.

3.3 An Empirical Evaluation of Web-Based Fingerprinting

Khademi et al. (2015) [1] analisaram os principais métodos de *Web Fingerprinting*, sendo um deles o Canvas *fingerprinting*. Os autores desenvolveram um método que emprega diversos atributos, incluindo o HTML5 Canvas. Com relação a esse atributo, os autores seguiram um padrão similar ao da literatura: renderização de textos. A diferença está no fato do texto ser renderizado 7 vezes e a imagem conter formas geométricas. A renderização do HTML5 Canvas está ilustrada na Figura 3.3.

Figura 3.3: Renderização no HTML5 Canvas [1].

O experimento de Khademi et al. (2015) [1] consiste em disponibilizar uma página *Web* na qual estava presente um *Web Fingerprinting*. Como resultado, foram coletados 1.523 *fingerprints*, sendo que 90,41% deles são únicos. No entanto, segundo os autores, o HTML5 Canvas colaborou com apenas 0,05% da unicidade, apresentando uma entropia de grau 3,21 *bits*. Segundo os autores, com base no cálculo da entropia para as diversas características obtidas, o HTML5 Canvas fica atrás das características na seguinte ordem: appVersion, userAgent, mimeTypes, JavaScript Fonts, system fonts e plugins [1]. No entanto, de acordo também com os autores, aplicando o algoritmo *greedy forward feature selection* (GFFS) para classificação das características utilizadas no método, o HTML5 Canvas está no conjunto das mais relevantes que podem ser usadas em um *Web Fingerprinting*.

Embora eficaz, o método dos autores utiliza diversos atributos relacionados à características de hardware, o que pode fazer o *fingerprint* mudar com mais facilidade. Empregar outros atributos, relacionados ao hardware, pode colaborar para fazer que com o *fingerprint* permaneça o mesmo por mais tempo.

3.4 Beauty and the Beast: Diverting modern web browsers

Laperdrix et al. (2016) [2] projetaram e avaliaram uma técnica de *Web Fingerprinting* a fim de comprovar que a mesma é capaz de ser usado para o bem, mas que também possui um lado obscuro. Dessa maneira, os autores exploraram a extração das características propostas por [6] (que ao todo são 10), com a adição de 7 atributos que surgiram em decorrência da evolução do HTML5 e de suas APIs. Um dos atributos utilizados é o HTML5 Canvas, mas no contexto 2D. Os autores renderizam um *emoji* através de caractere *Unicode* e duas vezes o mesmo pantograma, a fim de obter alta entropia a partir da renderização, conforme ilustrado na Figura 3.4.

Figura 3.4: Renderização do desenho no HTML5 Canvas [2].

Os autores demonstraram que o uso de *emojis* na renderização do HTML5 Canvas fornece informações mais discriminatórias do que o comum, visto que cada fabricante de *smartphone* e versão de sistema operacional possui seu próprio conjunto de *emojis*, o que resulta em diversas organizações dos pixels da imagem. Por ser um caractere *unicode*, *emojis* podem ser usados na *Web*.

Segundo os autores, o HTML5 Canvas está entre os 5 atributos mais influentes do seu método. Isso porque ele permite capturar a diversidade do sistema operacional renderizando o mesmo pantograma duas vezes, onde na primeira é ordenado ao navegador que use uma fonte inexistente (***Font probing***), o que resulta no uso de uma fonte de *fallback*[2]. Dependendo do sistema e das fontes instaladas no dispositivo, a fonte de *fallback* difere e permite determinar qual o sistema em uso [2]. Permite também detectar diferenças entre navegadores durante a renderização das imagens, mesmo que a combinação de navegador e sistema operacional sejam idênticas entre dispositivos distintos. Por ser um recurso do HTML5, o Canvas pode ser usado amplamente até mesmo em dispositivos móveis, visto que os navegadores para esses dispositivos são baseados em HTML5 e descartam o uso do Adobe Flash e de *plugins* [2].

Nos experimentos realizados foram coletados 118.000 *fingerprints*, sendo que 89,4% deles são únicos. Vale ressaltar que o método proposto não emprega apenas o HTML5 Canvas, mas sim um total de 17 atributos. Outro resultado pode ser observado: Dependendo de como o Canvas é implementado, mas características

[2]Fonte usada quando uma outra requisitada é inexistente.

relacionadas ao hardware do dispositivo podem ser extraídas. Esse é um dos pontos que o método proposto nesta pesquisa explora.

3.5 Picasso: Lightweight Device Class Fingerprinting for Web Clients

Bursztein et al. (2016) [5] projetaram, desenvolveram e avaliaram uma técnica de *Web Fingerprinting* para detectar e prevenir ataques automatizados oriundos de clientes maliciosos em lojas de aplicativos. O objetivo do trabalho foi distinguir clientes autênticos daqueles que são emulados ou automatizados. Diferentemente dos outros trabalhos, que identificam de forma única um dispositivo, o método de Bursztein et al. determina a classe do mesmo, composta pela tupla {*navegador Web*; *SO*}. Tal determinação é realizada por meio de desafios criados a partir do HTML5 Canvas. A arquitetura do mecanismo está ilustrada na Figura 3.5:

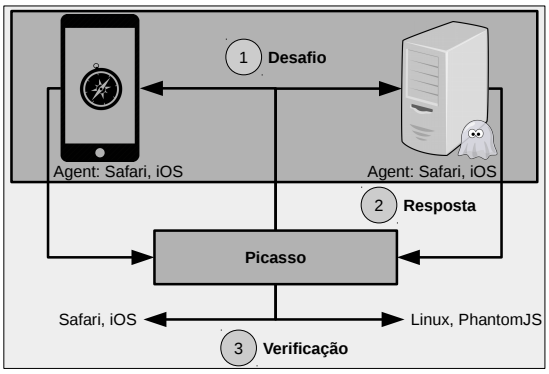


Figura 3.5: Arquitetura do Picasso em alto nível. (1) Um desafio é enviado para os clientes. (2) Os clientes respondem o desafio. (3) O Picasso verifica e distingue qual cliente é autêntico ou malicioso [5].

Os desafios são baseados em quatro operações de desenhos, chamadas de primitivas gráficas: *arc()*, *strokeText()*, *bezierCurveTo()*, *quadraticCurveTo()*. Segundo os autores, essas operações podem usar *anti-aliasing*[3] e algoritmos diversos para calcular ângulos e formatos de fontes. Além disso, cada elemento gráfico criado passa por um processo de customização com as seguintes operações: *createRadialGradient()*, *shadowBlur()*, *shadowColor()* para obter o máximo de entropia possível dos elementos gráficos. As primitivas, bem como todos os parâmetros das operações supracitadas, são escolhidas aleatoriamente, gerando diversos desafios

[3]Em processamento de imagem, é uma técnica usada para reduzir detalhes distorcidos na imagem.

[5]. Um deles a a comparação entre suas diferentes renderizações está ilustrado na Figura 3.6.

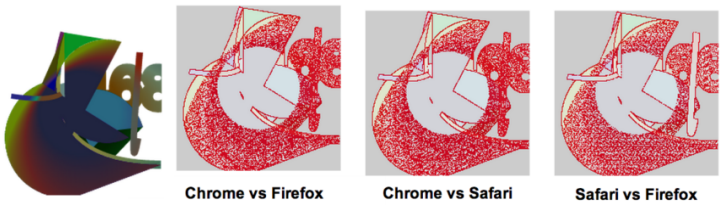


Figura 3.6: Comparação das renderizações do mesmo desafio (figura mais à esquerda) gerado pelo Picasso entre três navegadores *Web* distintos. As diferentes organizações dos *pixels* estão destacadas em vermelho [5].

Dois tipos de experimento foram realizados, um controlado e outro aberto. O ambiente controlado dispôs de uma amostra contendo 272.198 dispositivos. Já o experimento em cenário aberto possuiu participações anônimas de 52 milhões de usuários em um ambiente de produção. Em todos os casos, a ferramenta foi capaz de distinguir as classes dos dispositivos com 100% de acurácia. Esses resultados se aplicam para todas as combinações de navegadores e sistema operacionais (móveis e *desktops*) [5].

3.6 TARP Fingerprinting

Ximenes et al. (2016) [13] descreveram em seu trabalho técnicas usadas para projetar artefatos de *Web Fingerprinting* e apresentaram um método que utiliza o HTML5 Canvas para identificar, de forma única, um usuário. A abordagem proposta pelos autores foi baseada em trabalhos acadêmicos e bibliotecas de *Web Fingerprinting* existentes. Além disso, os autores descreveram contramedidas para combater o Canvas *fingerprinting* e implementaram seu método, de tal forma que o mesmo é capaz de detectar se o dispositivo possui ou não alguma contramedida instalada.

Diferente dos outros trabalhos, o método dos autores utiliza dois elementos HTML5 Canvas. O primeiro é utilizado para extrair a organização dos *pixels* da imagem renderizada (ilustrada na Figura 3.7) e, consequentemente, identificar o usuário; enquanto o segundo é utilizado como um detector de contramedidas instalas no dispositivo. Para a detecção, é utilizada uma imagem de baixa resolução no segundo Canvas, dessa forma, gerando um conjunto de chaves similares em todos os dispositivos, devido a sua baixa entropia. Com isso, é possível verificar se o segundo Canvas sempre retorna um *fingerprint* que está contido no conjunto de chaves identificadoras. Segundo Ximenes et al. (2016) [13], se o dispositivo

possuir alguma contramedida ao Canvas *fingerprinting*, o *fingerprint* obtido vai estar fora do conjunto de chaves predefinidas[4] para o segundo Canvas.

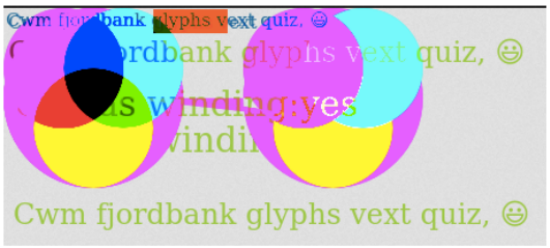


Figura 3.7: Exemplo da imagem gerada pelo TARP Fingerprinting [13]

Como resultado, é desenvolvido um artefato de *Web Fingerprinting*, baseado no Canvas, capaz de detectar se uma contramedida está sendo utilizada ou não no dispositivo do usuário. Ademais, foram obtidos 64.086 *fingerprints*, sendo que desse total 2.550 são distintos e 1.091 são únicos. O grau de entropia, segundo os autores, é de 7,86 bits. Como limitações, podem-se destacar:

- Com dois HTML5 Canvas sendo renderizados, mais recursos computacionais são exigidos (mas para o usuário é imperceptível);
- Os autores utilizam apenas o HTML5 Canvas para identificar o usuário. Segundo a literatura, usar apenas uma técnica reduz a eficácia do método de *Web Fingerprinting*.

3.7 Online Tracking: A 1-million-site Measurement and Analysis

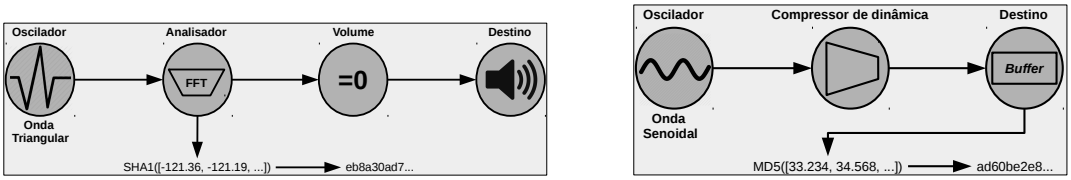
Englehardt e Narayanan (2016) [15] desenvolveram uma ferramenta chamada Open WPM para analisar os *sites* que estão no top 1 milhão do Alexa.com. Essa análise objetivou investigar *sites* que rastreiam seus usuários, seja por métodos que guardam estado (*cookies*) ou não (*fingerprinting*). O estudo descreveu as vantagens dos resultados descobertos com a ajuda do OpenWPM em relação aos trabalhos anteriores, pois a ferramenta permitiu que navegadores mais usuais fossem utilizados no processo de coleta. Além disso, novos métodos de rastreamento e de obtenção de identificadores foram identificados.

Dentre os novos métodos de rastreamento encontrados, Englehardt e Narayanan (2016) [15] destacam métodos de *Web Fingerprinting* que empregam o *AudioContext* API, *Battery* API e WebRTC (*Web Real-Time Communication*). Com

[4] As chaves identificadoras são predefinidas de acordo com experimentos empíricos, com a utilização de diversos dispositivos.

relação aos métodos que guardam o estado, o estudo mostrou que muitos *sites* utilizam o *cookie sync*[5], permitindo que *cookies* removidos pelo usuário sejam restaurados. Além disso, alguns métodos do HTML5 permitem restaurar esses identificadores de forma mais transparente do ponto de vista do usuário, com o uso do *localStorage* do HTML5.

Um dos pontos crucias apresentados por Englehardt e Narayanan (2016) [15] foi a descoberta do *AudioContext* em métodos de *Web Fingerprinting* para rastrear usuários. Os autores estudaram dois métodos e demonstraram a dinâmica dos mesmos, como ilustrado na Figura 3.8.



(a) *Analyser* (b) *Dynamics Compressor*

Figura 3.8: Análise de dois AudioContext *fingerprinting* [15].

O método da Figura 3.8a utiliza uma onda triangular (sinal) gerada a partir do *OscillatorNode*, transmitindo-a para o *AnalyserNode*. Esse nó provê as frequências do sinal e acesso à FFT do mesmo. Depois o sinal passa pelo *GainNode* para ter o volume ajustado para 0 e, por fim, é transmitido para caixa de som do dispositivo. Vale ressaltar que um *scriptProcessorNode* é utilizado para processar o áudio, obtendo sua FFT e gerando um *hash* a partir da mesma, que por sua vez é utilizado como *fingerprint* [15]. Ao lado, na abordagem ilustrada pela Figura 3.8b, a fonte de áudio é conectada a um *DynamicsCompressor*, provavelmente para obter diferenças no sinal processado entre máquinas diferentes. A saída do compressor é emitida para um *buffer* em um *OfflineAudioContext*. Dessa forma, as frequências do sinal contidas no *buffer* são somadas para gerar a *hash* e consequentemente o *fingerprint* [15].

Como resultado, Englehardt e Narayanan (2016) [15] projetaram um *AudioContext fingerprinting* com base nos métodos encontrados, obtendo 713 *fingerprints* distintos do total de 18.500, com entropia estimada em 5,4 *bits*.

Dessa forma, é possível notar que a Web Audio API fornece recursos que podem ser usados em Web Fingerprinting. Esse é um dos pontos que o método proposto nesta pesquisa explora.

[5]Compartilhamento de cookies entre terceiros.

3.8 Discussão

Este Capítulo apresentou algumas técnicas de *Web Fingerprinting* propostas na literatura e incluídas como pesquisas correlatas desta dissertação. A Tabela 3.1 sumariza os métodos e trabalhos apresentados, onde estão listados os autores e o ano de publicação (linhas), bem como as métricas que foram observadas durante o estudo dos trabalhos relacionados (colunas). Para tornar a tabela mais compacta, algumas palavras estão abreviadas, sendo elas: fontes (Ft) e caracteres (Ctr). Além disso, a palavra *Desenhos* faz referência a formas geométricas, retas, curvas, por exemplo, e a palavra *Efeitos* faz referência a customização, por exemplo, sombreamento, cor da fonte, transparência, dentre outros.

Tabela 3.1: Comparação dos métodos de Canvas *fingerprinting*.

Autores x Características	Ft. nativas	Ft. *Web*	Ft. *fallback*	Desenhos 2D	Desenhos 3D	Ctr. *Unicode*	Efeitos
Mowery et. al 2012 [9]	X	X	X		X		X
Nakibly et. al 2015 [11]					X		X
Khademi et. al 2015 [1]	X		X	X			X
Laperdrix et. al 2016 [2]	X		X			X	X
Bursztein et. al 2016 [5]				X			X
Ximenes et. al 2016 [13]	X		X	X		X	X

Vale ressaltar que a Tabela 3.1 e esta seção são restritas ao Canvas *fingerprinting*, pois até agora não existe investigação detalhada sobre o *AudioContext fingerprinting* e, consequentemente, poucos métodos propostos (até agora tem-se apenas o conhecimento do experimento realizado por [15]).

Embora o *Web Fingerprinting* proposto por Mowery e Shacham (2012) [9] tenha sido o primeiro estudo relevante sobre o Canvas *fingerprinting*, o método proposto emprega apenas uma característica robusta. O ideal seria extrair o máximo de características discriminatórias do dispositivo a fim de identifica-lo unicamente [7]. Diferente do trabalho de Mowery e Shacham (2012), Laperdrix et al. (2016) [2] e Khademi et al. (2015) [1] combinam o Canvas *fingerprinting* com outras características relacionadas ao navegador e dispositivo, mostrando-se mais eficaz do ponto de vista da unicidade. O Canvas *fingerprinting* de Laperdrix et al. (2016) faz a adição de caracteres *Unicode*, que é dependente do sistema operacional instalado no dispositivo e da fabricante. Já Khademi et al. (2015) empregar 78 atributos dos mais variados. O problema com esses dois trabalhos é justamente a utilização de atributos mutáveis ou que podem ser forjados com o

uso de extensões (*userAgent*, lista de *plugins*, lista de fontes, atributos do Adobe Flash, dentre outros). Além disso, atributos como o Flash e lista de *plugins* não funcionam em plataformas móveis, pois os navegadores *Web* desses dispositivos utilizam apenas os recursos disponíveis no HTML5.

Nakibly et al. (2015) [11] utilizam o HTML5 Canvas com o WebGL para renderizar várias animações em 3D complexas e obter o número de quadros *frames* são renderizados de acordo com um tempo predefinido. Assim, o método proposto difere-se dos outros porque o mesmo não extrai características oriundas da renderização de imagens, mas sim do *clock* da GPU e CPU do dispositivo, dessa forma, alcançando maior grau de entropia. Como desvantagem, o método consume muitos recursos, especialmente tempo de processamento, inviabilizando sua implantação em um ambiente real sem levantar suspeitas no usuário devido a demora. Além disso, o método pode sofrer interferência de outros processos que estão usando recursos da GPU e CPU. Em comparação com métodos propostos por Khademi et al. (2015), Laperdrix et al. (2016) e Mowery e Shacham (2012), o diferencial de explorar a GPU acaba sendo uma desvantagem pela demanda de recursos.

O método de Ximenes et al. (2016) [13] mostra-se bastante resiliente contra extensões e *plugins* que bloqueiam o Canvas *fingerprinting*, sendo esse um diferencial em relação aos outros trabalhos, visto que atualmente existem ferramentas de contramedidas. Além disso, a imagem renderizada em sua proposta é uma combinação de bibliotecas e outros métodos de Canvas *fingerprinting* encontrados na literatura, o que aumentam as chances de uma chave identificadora ser única. Assim como em [9], é possível que o método proposto por Ximenes et al. (2016) pudesse ser mais eficaz se o mesmo adotasse, pelo menos, mais um atributo para tentar alcançar maior unicidade do *fingerprint*. Similarmente aos outros trabalhos, Bursztein et al. (2016) [5] empregam o Canvas *fingerprinting*, mas com o objetivo de identificar classe dos dispositivos e, dessa maneira, distinguir clientes autênticos daqueles que são maliciosos. Em relação aos outros trabalhos, a vantagem da técnica proposta é não renderizar a mesma imagem para todos os clientes, mas sim figuras aleatórias, causando imprevisibilidade para um atacante, mesmo que ele tenha conhecimento do *script* de *fingerprinting*. No entanto, essa imprevisibilidade também é o seu ponto fraco na hora de (re)identificar o usuário em uma visita subsequente.

Embora cada uma das propostas possua suas vantagens e desvantagens, é possível observar que o HTML5 Canvas é uma característica muito relevante em um *Web Fingerprinting* devido aos vários navegadores darem suporte ao seu uso [19] e seus atributos dependerem fortemente do hardware e de camadas de software (*drivers* de vídeo, motor de renderização, dentre outros), tornado-o um fator discriminatório na identificação única de dispositivos e seus usuários. Outro

fator que o favorece muito é que o HTML5 está sendo adotado por dispositivos móveis, dessa forma, não há necessidade de depender dos *plugins* e nem do Adobe Flash, características que, em relação ao HTML5 Canvas são mutáveis [2, 7].

Diante do exposto, o trabalho proposto nesta dissertação se diferencia dos demais ao utilizar, em conjunto, Canvas e o OfflineAudioContext, e mais oito (08) atributos que apresentam características relacionadas a hardware. Isso permite que o método explore características oriundas da GPU e da placa de som do dispositivo. Os oito (08) atributos adicionais possuem a finalidade de aumentar as chances de gerar identificadores únicos, pois apenas o Canvas e o OfflineAudioContext não são suficientes para gerar *fingerprints* distintos a medida em que a base de chaves cresce. Os detalhes sobre a utilização do OfflineAudioContext e da adição de oito (08) atributos são apresentados nos Capítulos 4 e 5, respectivamente.

3.9 Considerações do Capítulo

Nesse Capítulo foram apresentados os trabalhos que são correlatos com esta pesquisa. Ao observar a literatura, contatou-se que há espaço para explorar novos métodos de *Web Fingerprinting*, por exemplo, um método que explora a *Web API*. A Web Audio API pode contribuir para enriquecer os métodos de *Web Fingerprinting* com novos atributos e novas formas de extrair características dos dispositivos.

Capítulo 4

Avaliação da Web Audio API e HTML5 Canvas

Este Capítulo apresenta a avaliação do HTML5 Canvas e da Web Audio API (AudioContext), atributos alvo desta dissertação, com o intuito de definir como usa-los nesta pesquisa. Os artefatos desenvolvidos e os resultados alcançados neste Capítulo são oriundos de experimentos empíricos realizados em ambiente controlado e aberto. Primeiro são apresentados os resultados de Canvas e em seguida do AudioContext.

4.1 Canvas *fingerprinting*

Conforme já apresentado nos Capítulos anteriores, o Canvas é um importante elemento para geração de um *fingerprint* e bastante pesquisado e implementado. Dessa forma, optou-se nesta dissertação por não se criar um novo método Canvas e sim por se utilizar uma ou mais implementações disponíveis na literatura. Foram avaliadas as técnicas propostas por Bursztein et al. [5], Khademi et al. [1], Laperdrix et al. [2], Mowery e Shacham [9], Nakibly et al. [11] e Ximenes et al. [13]. Para fazer a seleção, as técnicas propostas na literatura foram comparadas em experimentos empíricos com usuários voluntários (Figura 4.1).

O experimento consiste em embarcar os *scripts* de Canvas *fingerprinting* em uma página *Web* (1), que é disponibilizada para os usuários visitarem. Quando a página é requisitada (2) e carregada, os *scripts* são executados, extraindo as características dos dispositivos dos usuários, convertendo-as em um *hash* e armazenando-as no banco de dados (3). Optou-se por reunir todos os métodos em um só *script* porque foi constatado em um experimento anterior que separar um método por *script* pode acarretar em experimentos incompletos. Por exemplo, devido a problemas de conexão, apenas dois dos cinco métodos poderiam ser

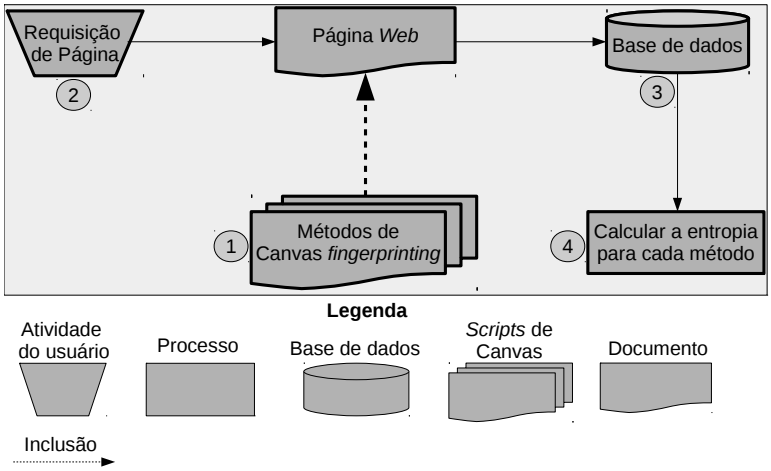


Figura 4.1: Esquema do experimento para selecionar um método de Canvas *fingerprinting*.

executados, causando inconsistência na base de dados, o que afetaria a análise dos resultados. Por último, é calculada a entropia de Shannon das amostras para cada método de *Web Fingerprinting* (4), a fim de determinar os maiores graus.

Para preparar o experimento do HTM5 Canvas, foram comparados quatro (04) métodos de Canvas *fingerprinting* propostos na literatura (Mowery e Shacham [9], Laperdrix et al. [2], Ximenes et al. [13] e Khademi et al. [1]). Os outros métodos que constam nos trabalhos correlatos não foram considerados devido a certas restrições. Por exemplo, o método proposto por Nakibly et al. [11] é muito demorado para executar e sobrecarrega o dispositivo e a proposta de Bursztein et al. [5] tem como objetivo determinar a classe de um dispositivo, não identifica-lo de forma única.

Para armazenar as características e outros campos obtidos, utilizou-se um banco de dados com uma tabela para cada método de Canvas *fingerprinting*. Dessa forma, foi possível calcular para cada um deles o grau de entropia das amostras de *fingerprint* e, consequentemente, selecionar o(s) método(s) mais eficaz(es) que se adequam a esta pesquisa. Quanto maior a entropia, mais distintas são as amostras e, consequentemente, maior o número de dispositivos identificados unicamente. Em contrapartida, quanto menor for a entropia, menos distintas são as amostras e, consequentemente, menor o número de dispositivos identificados unicamente.

Embora a decisão de projeto explanada acima seja favorável, existe o problema de um dispositivo acessar a página mais de uma vez e assim gerar repetições de chaves identificadoras. Para resolver isso, um *cookie* é instanciado no dispositivo do cliente apenas para sinalizar que o mesmo visitou a página. Dessa maneira,

só é feito o *fingerprint* do dispositivo que está acessando a página pela primeira vez. Assim, se houver alguma repetição de *fingerprint*, será devido a uma chave associada a dois dispositivos distintos.

O experimento foi realizado entre os dias 12/03/2017 à 26/03/2017, totalizando 2 semanas. Para que pudesse ser realizado, o experimento contou a ajuda de voluntários. Dessa forma, foram obtidos 50 *fingerprints* usados para comparar os métodos de Canvas *fingerprinting* listados. A entropia dos *fingerprints* para cada método está ilustrada na Figura 4.2. Vale ressaltar que sigla *wf* se refere a *Web font* (característica empregada por um dos trabalhos relacionados). Decidiu-se fazer essa separação para averiguar se o uso de *Web fonts* contribui para um *fingerprint* com maior entropia.

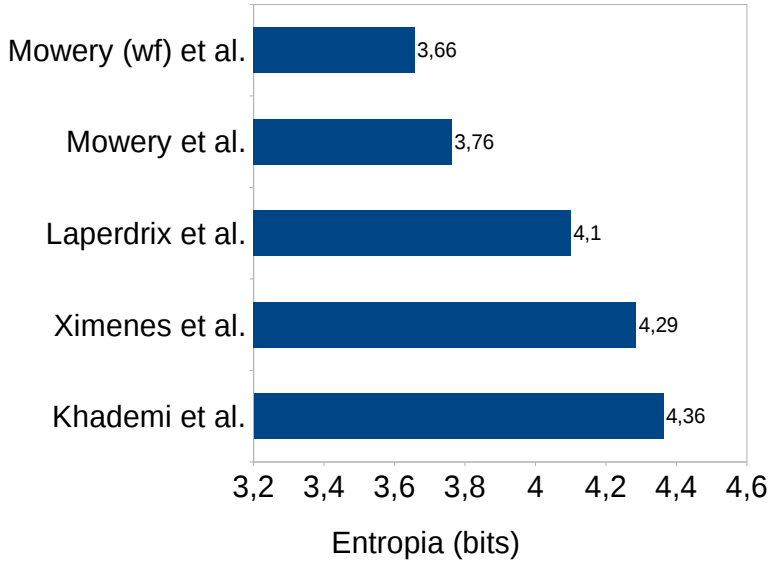


Figura 4.2: Resultado da comparação dos métodos de Canvas *fingerprinting*.

Os dois métodos de Canvas *fingerprinting* que mais se destacaram no experimento foram aqueles apresentados por Khademi et al. e Ximenes et al., respectivamente. No entanto, não foi possível obter o código do Canvas apresentado pelo segundo método. Dessa forma, decidiu-se utilizar o Canvas proposto por Laperdrix et al., no lugar daquele apresentado por Ximenes et al. Essa decisão foi tomada por três motivos. O primeiro é que foi possível obter de forma rápida os códigos de Canvas do Laperdrix et al. e de Khademi et al. O segundo motivo é que o Canvas proposto por Ximenes et al. se baseia no mesmos elementos encontrados no Canvas do Laperdrix et al. Por último e não menos importante, o Canvas do Laperdrix et al. é o terceiro que apresentou maior entropia.

A comparação entre os métodos de Canvas *fingerprinting* tem sua importância para o método proposto nesta dissertação porque o Canvas do método é baseado

nos dois métodos de Canvas que apresentam as maiores entropias encontrados nos trabalhos relacionados.

4.2 *AudioContext* vs *OfflineAudioContext*

Tendo em vista que a Web Audio API fornece duas interfaces (*AudioContext* e *OfflineAudioContext*) para trabalhar com processamento de áudio, é necessário averiguar aquela que apresenta maior precisão na hora de coletar os dados para gerar o *fingerprint*. Dessa forma, foram realizados seis (6) experimentos, ambos empregando diferentes nós de áudio. Um resumo dos experimentos é apresentado na Tabela 4.1, seguido da explicação de cada um mais adiante.

Tabela 4.1: Cenários dos experimentos.

Cenário	**Conjunto de *AudioNodes***	**Objetivo**
CE1	Recurso de áudio + *AudioContext* + *Analyser*	As frequências obtidas por meio das funções do *AnalyserNode* são estáveis, considerando um áudio no formato mp3 ou ogg?
CE2	Recurso de áudio + *OfflineAudioContext*	As frequências obtidas por meio do *AudioBuffer* do *OfflineAudioContext* são estáveis, considerando um áudio no formato mp3 ou ogg?
CE3	Sinal + *OfflineAudioContext*	As frequências obtidas por meio do *AudioBuffer* do *OfflineAudioContext* são estáveis, considerando as ondas periódicas?
CE4	Sinal + *OfflineAudioContext* + *Analyser*	Depois de obter as frequências estáveis por meio do *AudioBuffer* do *OfflineAudioContext*, as mesmas vão continuar estáveis após serem reobtidas por meio das funções do *AnalyserNode*?
CE5	Sinal + *AudioContext* + *Analyser*	As frequências obtidas por meio das funções do *AnalyserNode* são estáveis, considerando as ondas periódicas?
CE6	Sinal + *ScriptProcessorNode*	As frequências obtidas por meio do *AudioBuffer* de entrada do *ScriptProcessorNode* são estáveis, considerando as ondas periódicas?

Todos os experimentos foram realizados em ambiente controlado com um número de dispositivos limitado, sendo 1 notebook (Linux/Ubuntu 16.04 LTS), 1 desktop (Windows 10), 1 smartphone (Android 6) e uma máquina virtual[1] (Mac OS X 10.11). O processo de experimentação ocorreu da seguinte maneira: A página do experimento[2] contendo o *script* de *fingerprinting* que emprega a *Web Audio API* foi acessada por todos os navegadores em cada um dos dispositivos. Em cada acesso, foram testados os quatro tipos de ondas periódicas (senoidal, quadrada, triangular e customizada, como ilustra a Figura 4.3), com intervalo de 5 segundos entre cada onda e de 25 segundos para se iniciar um novo experimento. Esse acesso (experimento) à página foi realizado 50 vezes para cada navegador.

[1]Devido a falta de disponibilidade de uma máquina física.
[2]Por ser um experimento controlado, a página ficou hospedada em um servidor local.

Os navegadores utilizados foram: Chrome 59, Firefox 54, Edge 15, Safari 10 e Opera 46.

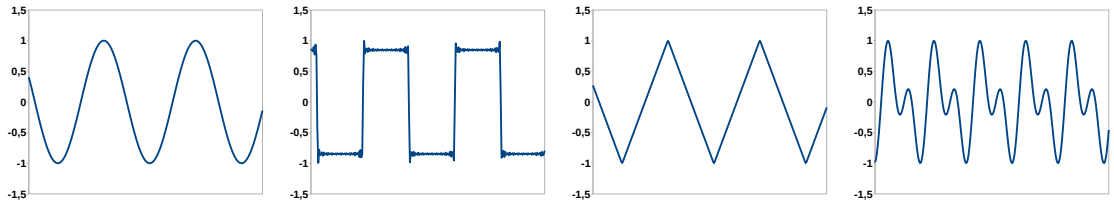


Figura 4.3: Ondas periódicas. Da esquerda para a direita: Senoidal, Quadrada, Triangular e Customizada.

O primeiro experimento (CE1) consistiu em carregar um recurso de áudio no navegador, decodificá-lo, renderizá-lo no dispositivo e então armazenar todos os *AudioBuffers* em uma lista. Ao final, o *AudioBuffer* é recuperado e enviado para o servidor, onde será calculado o *fingerprint* com o emprego da função de *hashing* SHA512. O segundo experimento (CE2) consistiu em carregar um recurso de áudio no navegador, decodificá-lo, enviá-lo para um *AudioBuffer* e, por fim, renderizá-lo no dispositivo, fazendo com que o som fosse emitido. Embora similar ao primeiro experimento, as frequências do *AudioBuffer* foram alimentadas pelo *OfflineAudioContext* e não pelo *AudioContext*. A Figura 4.4 ilustra os resultado dos experimentos CE1 e CE2.

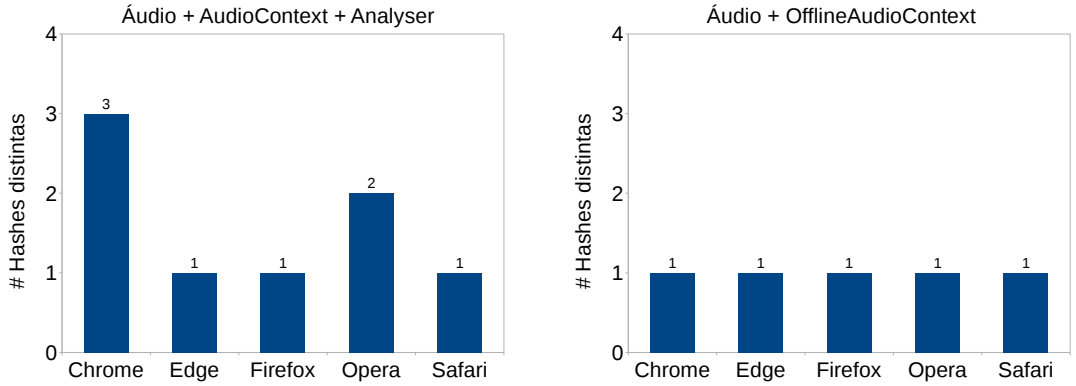


Figura 4.4: Experimentos com os cenários CE1 e CE2.

Percebe-se no primeiro cenário (Figura 4.4 à esquerda) que o conjunto de frequências (resultante da FFT) não se mantém estável durante as repetições, refletindo em *hashes* diferentes para o mesmo navegador. O navegador Chrome, por exemplo, gerou 3 *hashes* diferentes. Tal fato, demonstra a inviabilidade do

uso destes elementos (componentes) na geração de um *fingerprint*. Já o cenário CE2 sempre gerou um único *hash* para cada navegador (Figura 4.4 à direita). Isso serve de indicativo que o *OfflineAudioContext* pode contribuir com artefatos de *Web Fingerprinting*.

O experimento CE3 (Figura 4.5 à esquerda) consistiu em gerar três ondas primárias e uma customizada em um *AudioBuffer* do *OfflineAudioContext*. Depois de geradas, um conjunto com 2048 frequências de cada onda periódica foi obtido, enviado para o servidor e, por fim, o *fingerprint* foi calculado com base no conjunto de frequências obtido. O experimento CE4 (Figura 4.5 à direita) consistiu em renderizar, diretamente no dispositivo do usuário, as ondas periódicas. Da mesma forma que no experimento CE3, um conjunto com 2048 frequências de cada onda foi obtido e então o *fingerprinting* foi calculado.

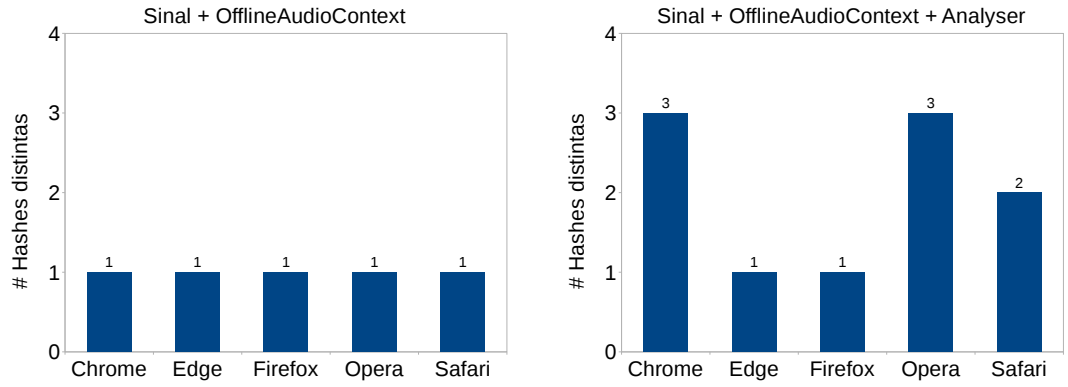


Figura 4.5: Experimentos com os cenários CE3 e CE4.

Percebe-se que no experimento CE3 todos os navegadores geram a mesma chave no decorrer das repetições. Tal comportamento é um indicativo de que o *OfflineAudioContext* pode ser empregado em métodos de *Web Fingerprinting*. Por outro lado, no experimento CE4, as frequências não se mantém as mesmas no correr de diversas repetições. Os navegadores Opera e Chrome, por exemplo, geraram 3 chaves distintas ao longo das 50 repetições, colocando em evidência que empregar o *AnalyserNode* para obter as frequências resulta em instabilidade, mesmo que a fonte de onde são obtidas seja estável.

O experimento CE5 consiste em renderizar, diretamente no dispositivo do usuário, três ondas primárias e uma customizada. Durante a renderização, suas frequências são obtidas a partir das funções providas pelo *AnalyserNode*. Após a extração, as ondas são enviadas para o servidor, local onde o *fingerprint* é calculado a partir das frequências extraídas. Esse experimento é similar ao CE1, mas a diferença é que no lugar de um recurso de áudio, é um sinal que está sendo renderizado. O experimento CE6, similarmente ao CE5, consiste em renderizar

três ondas primárias e uma customizada. No entanto, durante a renderização, as frequências são obtidas por meio do *AudioBuffer* de entrada do *ScriptProcessorNode*.

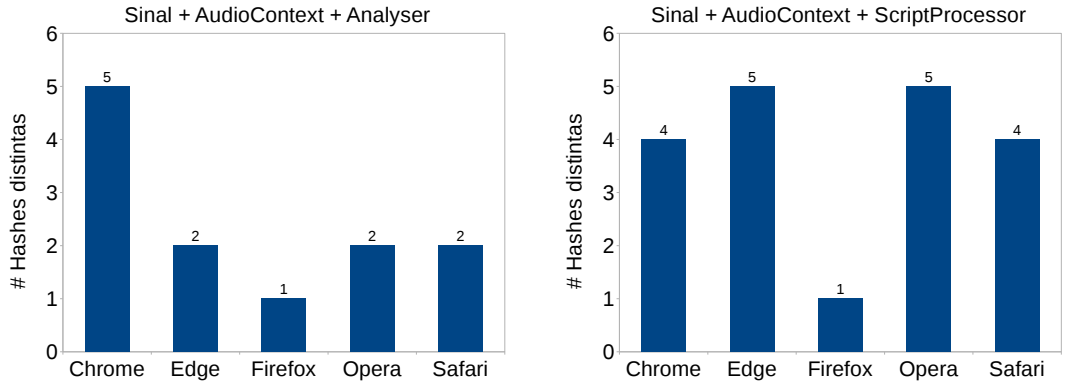


Figura 4.6: Experimentos com os cenários CE5 e CE6.

Na análise dos resultados, constatou-se que tanto no experimento CE5 quanto no experimento CE6 os conjuntos de frequências são voláteis, como ilustrado na Figura 4.6 à esquerda e direita, respectivamente. Por exemplo, na Figura 4.6 à esquerda, o Chrome apresentou 5 chaves distintas ao longo das 50 repetições. Na Figura 4.6 à direita, o Edge e o Opera apresentaram 5 chaves distintas. Dessa maneira, é possível observar que empregar o AudioContext e seus elementos não é recomendado para implementar métodos de *Web Fingerprinting*.

Analisando os resultados obtidos por meio dos experimentos CE1 até CE6, pode-se perceber que o *AudioContext* e os nós oriundos do mesmo (por exemplo, o *AnalyserNode*) não demonstram contribuir na obtenção de *fingerprints* estáveis, provavelmente pelo fato do *AudioContext* trabalhar em tempo real, onde latências adicionais devido a processamentos podem ocasionar instabilidades [27]. Por outro lado, o *OfflineAudioContext* mostra-se promissor e, por tanto, decidiu-se utilizar essa interface na implementação do *Web Fingerprinting* proposto neste trabalho. A investigação sobre o comportamento instável do *AudioContext* é um dos trabalhos futuros.

4.3 *OfflineAudioContext fingerprinting*

Esta seção apresenta os experimentos realizados com o *OfflineAudioContext* e os resultados alcançados, dado que essa interface mostrou-se promissora na subseção 4.2. Ao final, uma discussão sobre os resultados é apresentada na seção 4.4.

4.3.1 Experimentos com o *OfflineAudioContext*

Para comprovar a efetividade do *OfflineAudioContext*, um *script* que implementa o método proposto foi embarcado em uma página *Web*, que por sua vez foi hospedada em um servidor da Universidade, de forma que usuários pudessem participar do experimento. Em linhas gerais, quando a página é acessada, mostra-se uma mensagem explicando rapidamente o experimento e o propósito do mesmo. Além disso, um pequeno formulário deve ser preenchido com os seguintes campos: email, dispositivo (Desktop/Notebook ou Smartphone/Tablet, SmartTV, Smartwatch, Game console), e navegador (Chrome, Firefox, Opera, Safari, Internet Explorer, Edge, Chromium e outros). Esses dados informados pelos usuários tem como finalidade auxiliar na análise dos resultados. Além desses dados, o User Agent[3] também é coletado. Com relação às ondas periódicas, são empregadas as mesmas descritas na Subseção 4.2, ilustradas na Figura 4.3.

O uso de *OfflineAudioContext* nesta dissertação está fundamentado no fato de que navegadores e dispositivos distintos geram a mesma onda, mas com pequenas diferenças no conjunto de frequências [15]. Essas diferenças podem ser usadas para identificar a classe dos dispositivos, que por sua vez é representada pelo conjunto *{Navegador; Versão do navegador; Tipo do dispositivo}*. As frequências de todas as ondas periódicas são concatenadas e então são passadas por parâmetro para a função de *hashing* SHA512, gerando a chave que caracteriza uma classe de dispositivo. A dinâmica de funcionamento desse experimento é similar ao da Subseção 4.2. A cada experimento foram geradas ondas periódicas, com intervalo de 5 segundos (com exceção da primeira). Logo depois, um subconjunto contendo 2048 amostras de suas respectivas frequências foi obtido para cada onda. Após mais 5 segundos, os conjuntos foram enviados para o servidor, onde ficaram armazenados para análises posteriores. O experimento possuiu duração aproximada de 20 segundos para cada participante. Vale ressaltar que o intervalo de tempo para gerar as ondas, bem como o número de amostras, foram escolhidos de forma empírica e nenhum áudio foi emitido para os participantes.

4.3.2 Resultados e Discussão

A Figura 4.7 ilustra algumas características dos dispositivos das pessoas que participaram do experimento, realizado por 11 dias (16 à 26 de junho de 2017).

Foram registrados 129 participantes, contabilizando 155 acessos. No que diz respeito aos dispositivos, a maioria dos participantes fez uso de *desktops* ou notebooks (106 acessos contra apenas 45 de *Smartphones* e *Tablets*). Entretanto, mais à frente essa definição espontânea de qual tipo de dispositivo apresentará

[3] O User Agent é usado apenas para fins de validação do formulário preenchido pelo usuário, ou seja, não é utilizado pelo método proposto.

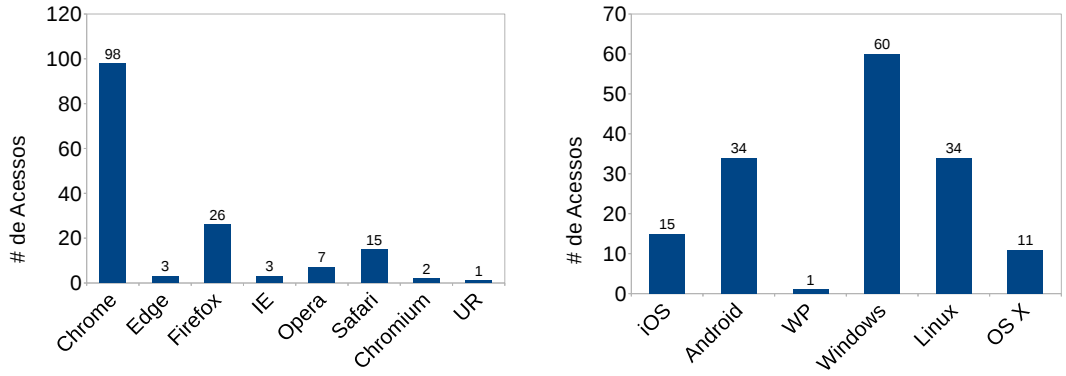


Figura 4.7: Características dos dispositivos usados no experimento.

alguns equívocos. Já sobre o navegador, o Chrome se mostrou o mais usado, seguido pelo Firefox e Safari (Figura 8 à esquerda). Navegadores menos usados como Chromium e UR também foram percebidos. Outro ponto interessante é a presença dos dois navegadores da Microsoft (Internet Explorer e Edge). Por fim, em relação aos sistemas operacionais, o Windows foi a plataforma mais usada, seguido pelo Android e Linux (empatados), iOS e OS X (Figura 4.7 à direita). Também foi detectado um Windows Phone (WP). Ao analisar as chaves identificadoras geradas, percebeu-se um fato interessante. Houve apenas 16 *fingerprints*, variando de acordo com o motor de renderização do navegador e com o tipo de dispositivo. Esse comportamento é descrito na Tabela do apêndice 8.3. As Tabelas abaixo apresentam resultados específicos dos experimentos.

A primeira observação que se faz na Tabela 4.2 é que os navegadores que possuem o mesmo motor de renderização compartilham a mesma chave. Vale ressaltar que optou-se por realizar esse agrupamento por se tratar do mesmo tipo de dispositivo (*Desktop*/*Notebook* no caso). Também percebe-se que a chave permanece constante, embora esses dispositivos apresentem versões distintas de navegador (Chrome nas versões 51 e 52 até 59, por exemplo), sistema operacional e versão do sistema operacional. Por outro ponto de vista, o uso do *fingerprinting* baseado no *OfflineAudioContext* permite saber qual motor de renderização está sendo utilizado, o que ajuda a caracterizar o dispositivo do usuário.

A segunda observação é que quando os navegadores possuem o mesmo motor de renderização e são da mesma versão, mas o tipo dos dispositivos em que estão executando é diferente (por exemplo, *Smartphone* e *Desktop*), as chaves geradas são distintas. Essa distinção pode ser empregada para determinar o tipo de dispositivo que está sendo usado pelo usuário, mesmo que haja extensões de contramedida que modifiquem, por exemplo, o User Agent, já que o processamento do sinal depende do motor de renderização do navegador e do dispositivo onde

Tabela 4.2: Chaves para os navegadores Chrome, Opera e Chromium.

Chave	Navegador	Versão	SO	Versão	Dispositivo
64e7af5ca...	Chrome	51, 54 - 59	Linux	–	Desktop/Notebook
64e7af5ca...	Chrome	58	Windows	10	Desktop/Notebook
64e7af5ca...	Opera	46	Linux	–	Desktop/Notebook
64e7af5ca...	Opera	45	Windows	10 e 7	Desktop/Notebook
64e7af5ca...	Chromium	53	Linux	–	Desktop/Notebook

o mesmo se encontra instalado. Esse resultado pode ser observado no navegador Chrome, versão 51 na Tabela 4.3.

Tabela 4.3: Chaves para os tipos distintos de dispositivos.

Chave	Navegador	Versão	SO	Versão	Dispositivo
64e7af5ca...	Chrome	51	Linux	–	Desktop/Notebook
749f38fac...	Chrome	51	Android	7	Smartphone/Tablet

A terceira observação diz respeito a versão do navegador afetar no *fingerprint* gerado. Por exemplo, percebe-se que o Firefox 44 e 47 (Tabela 4.4) possuem chaves diferentes, embora estejam instalados no mesmo tipo de dispositivo (*Desktop* ou *Notebook*) com o mesmo SO (Windows 7). Esse comportamento pode ser atribuído a alguma mudança no motor de renderização dos navegadores no decorrer das versões, bem como alguma nuance oriunda do *hardware* do dispositivo.

Tabela 4.4: Chaves para versões distintas do Firefox.

Chave	Navegador	Versão	SO	Versão	Dispositivo
412e24302...	Firefox	47	Windows	7	Desktop/Notebook
eed33e7c6...	Firefox	44	Windows	7	Smartphone/Tablet

Outro resultado interessante que foi observado durante a análise dos resultados é que os navegadores Safari, Chrome e Internet Explorer estão agrupados, mesmo que seus respectivos motores de renderização sejam distintos. Acontece que para esses navegadores o conjunto de frequências retornado foi nulo devido à alguma falha na rede, por falta de suporte do navegador à *Web Audio API* (algo que ocorre no Internet Explorer, console PS3 e em SmartTV com WebOS) ou por alguma proteção interna do próprio navegador. Esse comportamento pode ser atribuído ao fato da *Web Audio API* ainda não ter um padrão definido e também à limitação do método proposto.

A quinta e última observação é que o método proposto é capaz de identificar a classe dos dispositivos, ainda que informações falsas sobre os mesmos sejam passadas. Por exemplo, um Firefox que se passava por Safari foi agrupado com outros

Tabela 4.5: Navegadores em que o método não foi executado com sucesso.

Chave	Navegador	Versão	SO	Versão	Dispositivo
038e78e78...	Internet Explorer	11	Windows	10	Desktop/Notebook
038e78e78...	Internet Explorer	11	Windows	8.1	Smartphone/Tablet
038e78e78...	Safari	9	Windows	9	Smartphone/Tablet
038e78e78...	Safari	10	Windows	10	Smartphone/Tablet
038e78e78...	Safari	11	Windows	11	Smartphone/Tablet
038e78e78...	Chrome	59	iOS	10	Smartphone/Tablet
038e78e78...	Chrome	59	Linux	–	Desktop/Notebook

navegadores Firefox. O mesmo aconteceu com um dispositivo *Desktop/Notebook* que se passava por um *Smartphone/Tablet*, mas que foi agrupado com outros dispositivos do tipo *Desktop/Notebook*. Esses resultados foram confirmados com a verificação do user agent, embora essa característica não tenha sido empregada na geração do *fingerprint* para esse experimento.

4.4 Considerações do Capítulo

Os resultados apresentados neste Capítulo mostram que o OfflineAudioContext demonstra ser capaz de gerar a mesma chave identificadora no correr de diversas repetições, algo que é de muito importante, levando em consideração que um usuário pode acessar uma determinada página diversas vezes ao dia. O AudioContext, por outro lado, não demonstra ser capaz de gerar chaves consistentes no decorrer de diversas repetições, pelo menos não com a metodologia utilizada nesta pesquisa. Dessa maneira, decidiu-se utilizar o OfflineAudioContext como um dos atributos do método de *Web Fingerprinting* proposto nesta pesquisa.

Além do OfflineAudioContext, um método de Canvas será criado com base em dois métodos de Canvas entrados na literatura [1] e [2]. Embora o método de Ximenes et al. [13] tenha mais entropia do que o de Laperdrix et al. [2], o primeiro utiliza elementos do segundo, assim como o Canvas do método de *Web Fingerprinting* desenvolvido nesta pesquisa.

Capítulo 5

Proposta

Este Capítulo apresenta, como resposta à hipótese apresentada no Capítulo 1, o método proposto, explicando suas etapas e funcionamento. Vale ressaltar aqui que um método inicial foi proposto, mas após as limitações do AudioContext (explanadas e demonstradas no Capítulo 4), o método foi redefinido e o atributo OfflineAudioContext passou a ser utilizado. Antes de apresentar a solução, primeiramente serão apresentados os atributos escolhidos.

5.1 Atributos Empregados

O método proposto faz uso de dez (10) atributos relacionados ao hardware, em especial Canvas e Web Audio API. Como pode ser observado na Tabela 5.1, cada atributo contribui para a obtenção das caraterísticas dos dispositivos, a fim de gerar um *fingerprint*. Cada um dos atributos é explanado a seguir.

Tabela 5.1: Atributos utilizados pelo método proposto.

Atributo	**Descrição**
User Agent	Nome, versão e plataforma do navegador.
Platform	Plataforma para a qual o navegador foi compilado.
Resolution	Fornece as dimensões da tela do dispositivo
Waves	Ondas periódicas do *OfflineAudioContext*
Canvas	Contém o conteúdo do Canvas no formato de URL
OfflineAudioContext	Informações sobre o objeto do *OfflineAudioContext*
Hardware Concurrency	Número de processadores lógicos do dispositivo
WebGL Vendor	Nome da desenvolvedora do driver da GPU
WebGL Renderer	Informações sobre o driver da GPU
Media Devices	Enumera dispositivos de mídia

O **User Agent** contribui para informações sobre o sistema e o navegador que está sendo utilizado. Da forma que foi empregado, a versão de Sistema Operacional e de navegador não estão sendo utilizadas, com o objetivo de reduzir a mutabilidade do *fingerprint*, pois a cada atualização de navegador e, em alguns casos, de Sistema Operacional, o valor do User Agent é alterado [7].

O **Platform** fornece informações sobre a plataforma do Sistema operacional, por exemplo, alguns dos valores mais comuns são WIN32, WIN64, X11, etc. Esse dado contribui para obter mais características do dispositivo, principalmente quando há alguma discrepância[1] entre outros atributos já coletados[20].

O **Resolution** é a combinação de três atributos: largura, altura e profundidade de cores da tela. Embora não seja um atributo tão relevante para *notebooks* e *desktops*, tende a contribuir bastante quando trata-se de dispositivos móveis, já que há diversas dimensões de telas [20].

O **Waves** é uma onda periódica composta por outras quatro ondas: Senoidal, quadrada, triangular e customizada. As ondas periódicas exploram a capacidade de processamento de sinal do dispositivo, que por sua vez pode ser usado como atributo no momento de gerar um *fingerprint* para o usuário [15].

O **Canvas** é usado para renderizar desenhos e escrever textos com fontes e cores diferentes. O Canvas tem como entrada o atributo Waves, que ajuda a deixar a imagem variada entre dispositivos. Dessa maneira, o Canvas contribui para obter mais características com a ajuda do Waves, pois sua renderização não depende apenas do navegador da placa de vídeo, mas também de outros atributos [9]. Outras características (explicadas mais afrente) também são utilizados para compor o Canvas.

O **OfflineAudioContext**[2] é utilizado para obter informações padrões do objeto OfflineAudioContext, por exemplo, tamanho máximo do buffer, número máximo de canal, número de entrada e saída, estado, interpretação do canal, taxa de *samples*, número de canais e modo de contagem. Assim como o atributo waves, depende do hardware do dispositivo [15].

O **Hardware concurrency** fornece o número de processadores lógicos disponíveis no dispositivo para executar *threads* [23]. Esse atributo ajuda a diferenciar em nível de hardware os dispositivos, contribuindo para a diversificação das chaves geradas.

Os atributos **WebGL vendor** e **WebGL renderer** fornecem, respectivamente, o nome da desenvolvedora do driver da GPU e informações sobre o driver da GPU. Esses atributos contribuem para obter características relacionadas ao hardware. [28]

[1] Algumas extensões causam discrepâncias que servem como características para identificar usuários.

[2] Uma das interfaces da *Web Audio API*, como será explicado mais adiante.

O atributo **Media Devices** enumera os dispositivos de mídia encontrados no dispositivo, por exemplo, *webcams* e microfones. Dessa forma, é possível obter os nomes e modelos desses dispositivos, o que contribui para um *fingerprint* cujas características utilizadas são relacionadas ao hardware do dispositivo [23].

5.2 Método Proposto

A Figura 5.1 ilustra a esquematização do *Web Fingerprinting* proposto.

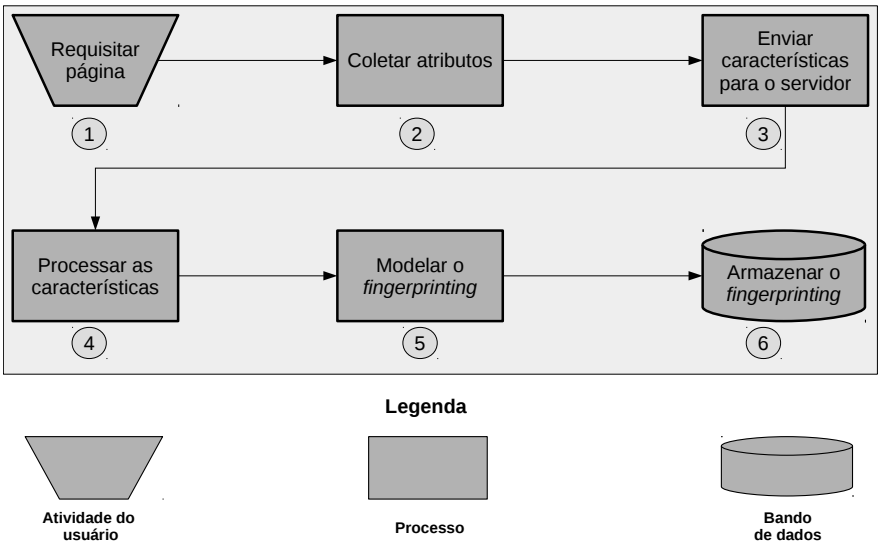


Figura 5.1: Esquematização da solução proposta.

É importante ressaltar que essa esquematização precisou ser ajustada após a constatação do problema/limitação do AudioContext (Capítulo 4) ao contribuir com a geração de um *fingerprint*. A principal diferença da proposta para a versão inicial planejada repousa no fato do uso de OfflineAudioContext ao invés do AudioContext e no seu emprego como entrada para o Canvas. A ideia foi utilizar as ondas periódicas geradas com o OfflineAudioContext, bem como os dados desta interface da *Web Audio API* e outros atributos (WebGL renderer, WebGL vendor e número lógicos de núcleos) como entradas para renderizar o Canvas. Além disso, outros atributos relacionados a características de *hardware* também são utilizados durante a renderização, como é explanado no Capitulo 6.

Pensando em aspectos de implementação, esta consiste em coletar os atributos (2,) sendo que essa etapa se divide em três, como ilustrada na Figura 5.2: coletar as ondas periódicas e os dados do objeto do OfflineAudioContext (2.1), desenhar o Canvas com as formas geométricas, pantogramas e as ondas periódicas (2.2),

coletar os outros atributos apresentados na Tabela 5.1 (2.3) e, por fim, utilizar o Media Devices, WebGL e Hardware Concurrency como entradas adicionais ao Canvas (2.4). Após essas sub-etapas, as características obtidas são enviadas para o servidor (3), onde são processadas (4) para então serem modeladas em um *fingerprint* capaz de identificar, de forma única, o usuário (5). Por fim, o *fingerprint* obtido é armazenado na base dados para análises posteriores (6).

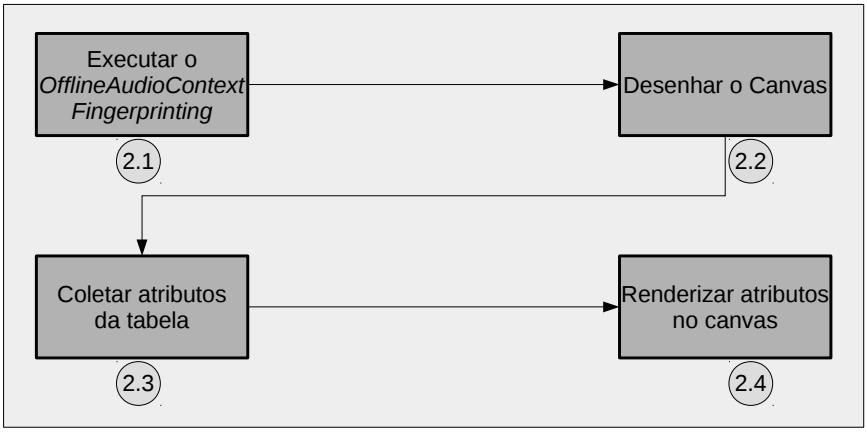


Figura 5.2: Sub-etapas da coleta de atributos (passo 2 da Figura 5.1).

No que tange as ondas periódicas, elas são geradas da seguinte maneira. Um contexto de áudio é criado a partir do OfflineAudioContext. Após a criação, os nós osciladores (responsáveis por gerar as ondas periódicas) são instanciados. Após os tipos de ondas estarem definidas, um nó que mescla canais é instanciado para mesclar as ondas em uma só. Após este processo, as frequências da onda resultante são obtidas a partir do AudioBuffer do OfflineAudioContext, resultando em um dos atributos utilizados pelo método proposto nesta pesquisa.

No que diz respeito à avaliação, o cenário é o seguinte: alguns dos métodos de *Web Fingerprinting* propostos nos trabalhos relacionados serão comparados com o proposto nesta dissertação. Dessa forma, a avaliação do método proposto leva em consideração todas as propriedades empregadas pelas técnicas de *Web Fingerprinting* dos trabalhos relacionados, a fim de avaliar o potencial da combinação de atributos fortemente relacionados ao hardware e analisar se existem vantagens significativas na geração do *fingerprint* a partir do mecanismo proposto quando o mesmo é confrontado com aqueles encontrados na literatura.

5.3 Funcionamento

Já foi explicado neste Capítulo que o método proposto emprega dez (10) atributos, sendo o Canvas e o OfflineAudioContext os mais importantes. Esta seção

explica o funcionamento do OfflineAudioContext no contexto do método proposto, bem como do Canvas.

5.3.1 OfflineAudioContext *fingerprinting*

Ao contrário do HTML5 Canvas, o OfflineAudioContext ainda não foi amplamente explorado na literatura, o que abre oportunidades para a investigação do potencial que pode ser desencadeado de um *Web Fingerprinting* que o emprega como atributo. Até o tempo de escrita desta dissertação, apenas Englehardt e Narayanan (2016) [15] relataram e realizaram experimentos de OfflineAudioContext *fingerprinting*.

A implementação do OfflineAudioContext segue a documentação da *Web Audio API* e consiste em: Criar um contexto de áudio (OfflineAudioContext) (1); Gerar fontes de áudio a partir do contexto (2); configurar o áudio (Criar os efeitos sonoros) (3); Criar um *ChannelMergerNode* (4); Selecionar o destino do áudio (5) e, por fim, obter o conjunto de suas frequências (características) (6) [15, 23]. Os passos (1) - (6) resultam em um sinal que não necessariamente é audível a partir da caixa de som, mas que é processado. Parte do processamento consiste em calcular a FFT (*Fast Fourier Transform*) do sinal, cujo resultado extraído é usado como característica do dispositivo do usuário, capaz de identificá-lo unicamente [15]. A Figura 5.3 ilustra, em alto nível, como é implementado o OfflineAudioContext *fingerprinting* que compõe o método de *Web Fingerprinting* desta dissertação.

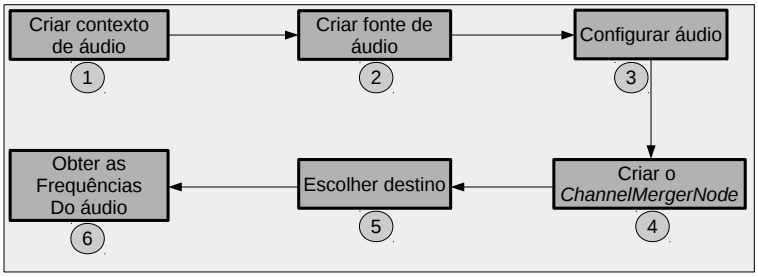


Figura 5.3: Representação em alto nível do AudioContext *OfflineAudioContext Fingerprinting*.

Na prática, quatro (04) tipos de ondas são utilizadas no OfflineAudioContext: Senoidal, Quadrada, Triangular e Customizada. No entanto, em vez de concatenar todos esses sinais, o método utiliza um *ChannelMerger* com quatro canais. Esse *AudioNode* faz a junção dos sinais que passam por cada canal, tornando-os um só.

A Figura 5.4 ilustra a onda gerada, onde cada uma das ondas entra em um canal específico e disso resulta uma nova onda periódica. A onda resultante possui frequência de oscilação no valor de 200 hHz, que é suportável pelos navegadores mais populares existentes. A ideia de usar o máximo da frequência é forçar o hardware a trabalhar mais, com o objetivo de extrair um sinal periódico de acordo com as características do dispositivo. O sinal resultante é então usado em duas fases. A primeira para compor o método de *Web Fingerprinting* como um todo.

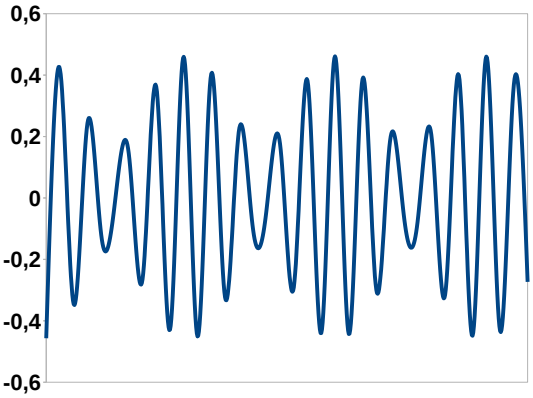


Figura 5.4: Sinal resultante da junção dos quatro tipos de ondas periódicas.

A segunda, para compor o Canvas, consiste em obter o *hash* MD5 da onda periódica e, dessa forma, os seis (06) primeiros caracteres que são utilizados para compor a cor do Canvas, que é usada para plotar a onda periódica no Canvas e outras características oriundas do OfflineAudioContext, tais como *sampleRate*, *channelCountMode*, *state* e *channelCount*. Além disso, umas das figuras geométricas do Canvas é contornada por essa cor.

5.4 Canvas *fingerprinting*

Além da onda periódica e dos dados oriundos do objeto do *OffilineAudioContext*, três outros atributos são utilizados para compor o Canvas: WebGL renderer, WebGL Vendor e número de processadores lógicos. Da mesma forma que o OfflineAudioContext, os *hash* desses atributos são utilizados para definir mais duas cores no Canvas: uma usada para as informações relacionadas ao WebGL *renderer* e *vendor* e a outra usada para renderizar o pantograma, *emoji* e o número de processadores lógicos.

Além disso, o Canvas é implementado de duas formas: a primeira é baseada nos trabalhos de Khademi et al. [1] e Laperdrix et al. [2]. Dessa forma, o Canvas

faz uso de formas geométricas, pantogramas, *emoji*, fonte falsa e padrão, para poder explorar características e hardware e, além disso, do navegador também. A segunda forma consiste na composição dos atributos explanados nas suas subseções anteriores. Dessa maneira, o Canvas não é formado estaticamente, mas sim de forma dinâmica por meio de outros atributos (sendo estes dependentes do hardware do dispositivo). Isso significa que, mesmo em dispositivos que possuam GPU iguais, mas capacidade de processamento de sinal distintas, dois Canvas diferentes serão gerados. Dessa forma, o Canvas varia mais, o que pode colaborar para maior entropia do método proposto. O resultado da renderização do Canvas composto pelos atributos supracitados está ilustrado na Figura 5.5.

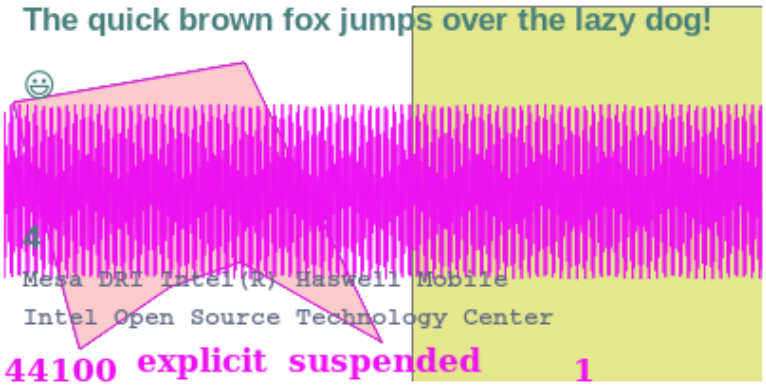


Figura 5.5: Canvas renderizado pelo método proposto.

5.5 Considerações do Capítulo

Este Capítulo apresentou a proposta, descreveu o fluxo de seu funcionamento e seus elementos. Os atributos apresentados na Tabela 5.1 foram escolhidos com o intuito de extrair características dos dispositivos por meio de tecnologias já consolidadas (o HTML5 Canvas, por exemplo) e a Web Audio API, tecnologia que ainda não foi largamente explorada em *Web Fingerprinting*. Dessa forma, o trabalho continua investimento um método de *Web Fingerprinting* que faz uso de tais recursos, para investigar se é possível, com novos avanços das tecnologias Web, desenvolver novas formas de gerar *fingerprints*.

Capítulo 6

Experimentos e Resultados

Este Capítulo apresenta os experimentos realizados sobre o método proposto. Embora tenham sido realizados diferentes experimentos entre os meses de agosto a novembro de 2017, sempre para ajustar o método, o experimento derradeiro foi realizado de 31/12/2017 até 23/02/2018 e ficou disponível publicamente para que qualquer pessoa pudesse participar. A dinâmica desse experimento foi similar àquela descrito no Capítulo 4 (subseção 4.3.1).

6.1 Resultado Quantitativo

No total, o experimento final contou com 145 participações, sendo que desse total 131 (90,34%) geraram chaves únicas, enquanto o restante (9,65%) resultaram em chaves duplicadas. Dispositivos com diversos navegadores fizeram parte do experimento. As Figuras 6.1, 6.2 e 6.3 ilustram os sistemas operacionais dos dispositivos, navegadores e os tipos de dispositivos detectados durante o experimento.

No total, foram descobertos (utilizados pelos participantes) seis (06) sistemas operacionais distintos durante o experimento. Foram percebidos 45 acessos feitos por Windows, 30 acessos feitos por Linux, 12 acessos por MacOS, 53 acessos por Android (53), 3 acessos por iOS e 2 acessos por Windows Phone. É importante ressaltar que as informações sobre os sistemas operacionais foram coletadas pelo atributo *User Agent*.

No que tange aos navegadores descobertos durante o experimento, foram percebidos 83 acessos oriundos de navegadores Google Chrome, 32 acesso vindos de Firefox, 5 acessos de Safari, 7 acessos de Opera, 1 acesso de Chromium, 9 acessos de Microsoft Edge e 8 acessos de navegadores não usuais como, por exemplo, UR[1]. Assim como os sistemas operacionais, os dados dos navegadores foram co-

[1]https://www.ur-browser.com/pt-BR/

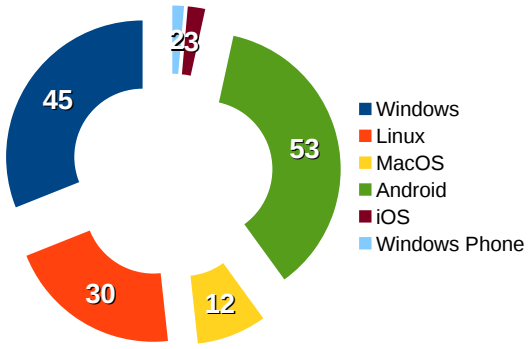


Figura 6.1: Sistemas Operacionais descoberto no experimento.

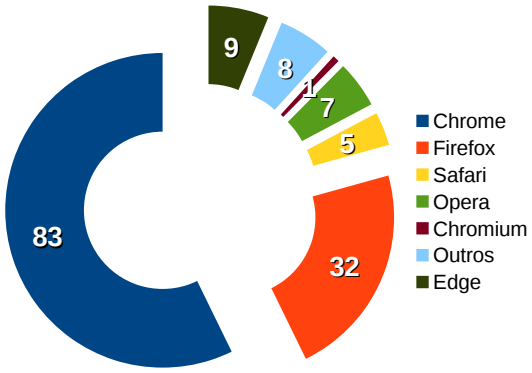


Figura 6.2: Navegadores utilizados no experimento.

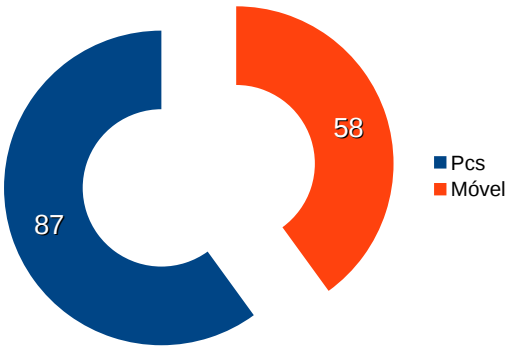


Figura 6.3: Tipos de dispositivos utilizados no experimento.

letadas pelo atributo *User Agent*. Por fim, em relação aos tipos de dispositivos descobertos, o experimento pedia que cada participante definisse como seria feito o acesso: se via um PC/Desktop ou via um dispositivo móvel. Foram registrados 87 acesso de PCs e 58 acessos de dispositivos móveis.

6.2 Relevância dos Atributos

Uma vez que esta dissertação se difere dos demais trabalhos encontrados na literatura por empregar atributos de hardware pouco utilizados e de forma não convencional, uma avaliação da relevância dos 10 atributos empregados se faz necessária, o que também serve como justificativa e prova para a unicidade dos 131 *fingerprints* obtidos com o experimento final. Dessa forma, todos os atributos coletados, dos 145 participantes, foram analisados, de acordo com sua relevância, através da entropia de Shannon e utilizando o InfoGain. A Tabela 6.1 mostra qual o grau de entropia de Shannon gerada para cada um dos atributos utilizados.

Tabela 6.1: Entropia de Shannon dos atributos utilizados pelo método proposto.

Atributo	**Entropia (*bits*)**
Canvas	6,6272
WebGL Renderer	5,6578
Media Devices	5,1292
User Agent	5,0963
Waves	3,41
WebGL Vendor	2,7747
Platform	2,6963
Resolution	2,0263
Hardware Concurrency	1,4726
OfflineAudioContext	0,6071

A Entropia de Shannon calcula o grau de imprevisibilidade, em outras palavras, o quanto algo é aleatório. No caso deste trabalho, o quanto mais imprevisível o conjunto de características e as chaves resultantes delas melhor, pois isso implica em uma base com chaves diversificadas.

Começando a análise pelo atributo Canvas, de acordo com a entropia, o método de Canvas *fingerprinting* implementado nessa dissertação tornou-se o atributo mais discriminatório do método. Isso pode ser justificado pela composição de atributos. Dos 145 Canvas obtidos, 87 são únicos e 23 se repetem um certo número de vezes, por exemplo, um dos Canvas se repetiu 4 vezes (1 ocorrência + 4 repetições). A Figura 6.4 ilustra a distribuição dos valores obtidos para o atributo Canvas.

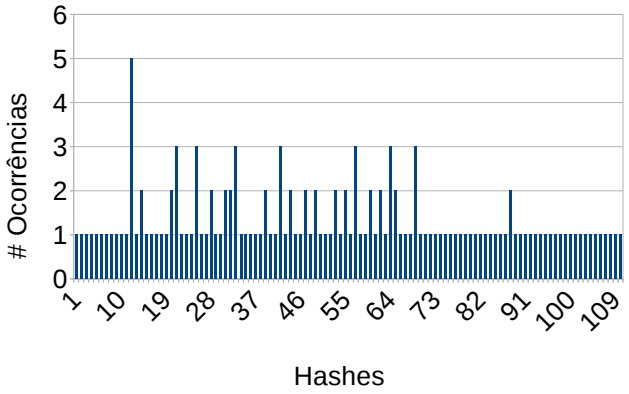


Figura 6.4: Distribuição dos valores do Canvas no método proposto.

Na Figura 6.4, as repetições no Canvas são causadas devido aos hardwares dos dispositivos serem similares a ponto de rendenrizarem o Canvas de forma idêntica. No entanto, vale notar que, embora o Canvas tenha repetições, não significa que o *fingerprint* final será duplicado.

Em segundo lugar, tem-se o atributo *WebGL renderer*, que provê informações sobre o driver da GPU. Pode-se notar que esse atributo, assim como o Canvas, também depende do hardware do dispositivo e mostra-se relevante para o método devido ao seu alto grau de entropia, se comparado com outros atributos empregados pelo método proposto nesta pesquisa. Dos 145 *WebGL renderer* obtidos, 33 são únicos e 32 são repetidos, mostrando que o *WebGL renderer* possui relevância, mas não tanto quanto o Canvas. Por exemplo, um dos WebGL se repetiu 8 vezes (1 ocorrência + 8 repetições). A Figura 6.5 ilustra a distribuição dos valores para o atributo *WebGL renderer*.

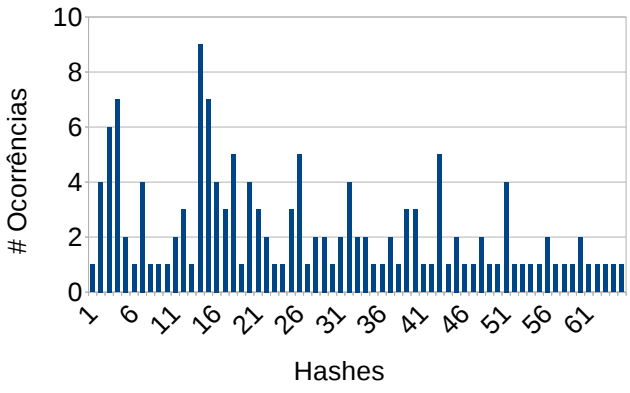


Figura 6.5: Distribuição dos valores do WebGL renderer.

Na Figura 6.5, as repetições podem ser explicadas pelo mesmo motivo encontrado no Canvas: hardware semelhante. Por exemplo, contatou-se que os dispositivos Moto G e Moto Z retornaram como resposta Adreno (TM) 506.

Em terceiro lugar pode-se notar que o atributo *Media Devices*, assim com os dois primeiros atributos, possui um grau de entropia alto. Vale ressaltar que dos 145 participantes, apenas 36 ocorrências desse atributo foram obtidas. Tal fato pode ser atribuído à falta de suporte dos navegadores ou simplesmente porque a permissão de acesso à esses dispositivos não foi concedida. Dessa forma, o cálculo da entropia para esse atributo desconsiderou as ocorrências nulas. A Figura 6.6 ilustra a distribuição dos valores para o atributo *Media Devices*, considerando os valores nulos. Na Figura é possível observar que, das 36 chaves, apenas 1 uma não é única. No entanto, foi repetida 109 vezes, o que prejudica a unicidade deste atributo.

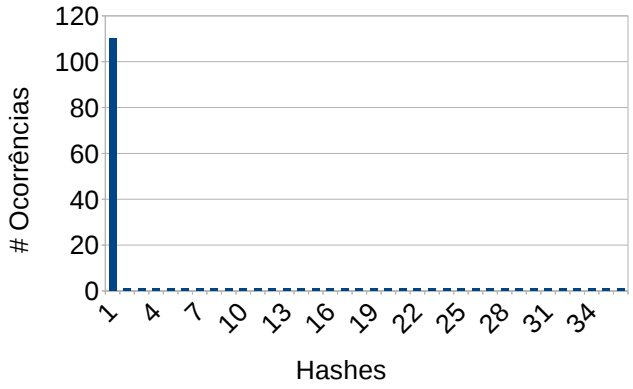


Figura 6.6: Distribuição dos valores do *Media Devices*.

Em quarto lugar está o atributo *User Agent*, que por sua vez fornece informações gerais sobre sistema operacional, nome e versão do navegador e, em alguns casos, o modelo do dispositivo. Sua alta entropia resido no fato de conseguir reunir esses quatro tipos de dados em um só lugar. No entanto, o método proposto não considerou números de versões, com o intuito de evitar mudanças no *User Agent* devido a atualizações. Dos 145 *User Agent* obtidos, 33 são únicos, enquanto 24 não o são. A Figura 6.7 ilustra a distribuição dos valores para o atributo *User Agent*. Como é possível observar na Figura, um dos valores do *User Agent* repete-se 18 vezes (1 ocorrência + 18 repetições).

Em quinto lugar ficou o atributo *Waves*, que são as ondas periódicas geradas no dispositivo. Esse atributou ficou com entropia próxima da média[2], que é 2,92, mesmo sendo um dos atributos que extrai características de hardware. Visto

[2]Foi calculada a média geométrica.

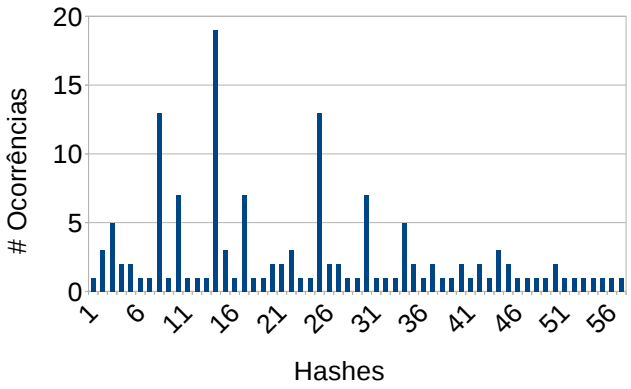


Figura 6.7: Distribuição dos valores do User Agent.

de forma isolada, ele não causa impacto no *fingerprint*, mas olhando para o Canvas do método proposto e comparando-o com os outros dois dos trabalhos relacionados, é possível notar que o *Waves* contribuiu com o método. A Figura 6.8 ilustra a distribuição dos valores para o atributo *Waves*. Como é possível observar na Figura, dos 145 valores de *Waves*, apenas 3 são únicos enquanto 14 não o são. Por exemplo, um dos valores para o atributos Waves repete-se 32 vezes (1 ocorrência + 32 repetições).

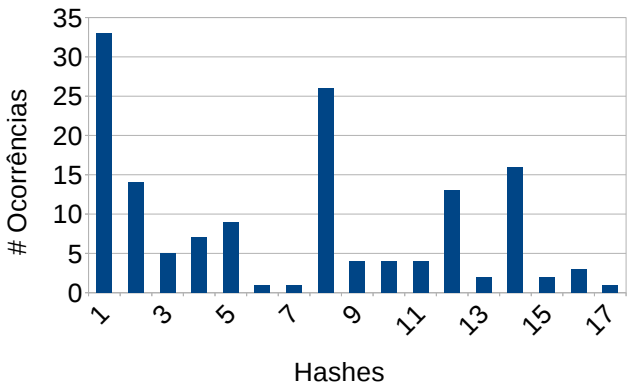


Figura 6.8: Distribuição dos valores do *Waves*.

Observando a Figura 6.8, é possível constatar que a placa de som não apresenta diferenças significas entre dispositivos, visto que para muitas delas, as ondas períodicas obtidas possuem o mesmo conjunto de frequências.

No *ranking* dos atributos restantes, tem-se o *WebGL vendor*, *Platform*, *Resolution*, *Hardware Concurrency* e *OfflineAudioContext*. Esses atributos não apresentaram variações significantes em seus valores, o que causou baixa entropia.

No entanto, para explorar os atributos que provêm informações sobre o hardware do dispositivo, estes foram usados como entradas para atributos mais fortes, com o objetivo de maximizar o grau de entropia do atributo mais forte, como foi observado com o Canvas. Para melhor visualização, a Tabela 6.2 apresenta um exemplo do conteúdo para cada um dos atributos utilizado pelo método.

Tabela 6.2: Exemplo de conteúdo para cada atributo.

Atributo	**Exemplo de conteúdo**
User Agent	Mozilla/. (Linux; Android...
Platform	Linux armv7l, MacIntel, ...
Resolution	360x640x32, 412x732x32, ...
Waves	-0.3022688031, 0.2772963...
Canvas	iVBORw0KGgoAAAA...
OfflineAudioContext	length:44100;sampleRate:44100...
Hardware Concurrency	8, 6, 4, 2,...
WebGL Renderer	Adreno (TM) 505, Apple A11 GPU, ...
WebGL Vendor	Qualcomm, ARM, NVIDIA, ...
Media Devices	videoinput: Camera 0; Facing back...

Além da Entropia de Shannon, outro método foi utilizado para avaliar a relevância dos atributos. O método empregado foi o InfoGain, onde foi necessário converter a base de dados para o formato que o Weka entende e, além disso, aplicar um filtro (*StringToNominal*) nos atributos e seus respectivos valores, para que a ferramenta pudesse processar os dados coletados. Após esse processo, foi possível obter o *ranking* de relevância dos atributos, como pode ser visto na Tabela 6.3.

É possível notar na Tabela 6.3 que o Canvas do método proposto continua sendo o atributo mais relevante. No entanto, vale ressaltar que esse Canvas é composto por cinco atributos relacionados ao hardware do dispositivo: *OfflineAudioContext* (propriedades e ondas periódicas), número de processadores lógicos e informações (versão do driver e desenvolvedora) do WebGL. Também é possível observar que as ondas periódicas (assim como outros atributos utilizados pelo método) possuem baixa relevância se comparadas aos Canvas dos trabalhos relacionados. Isso pode significar que, esses atributos (*OfflineAudioContext*, *Web GL vendor*, por exemplo) isoladamente não são capazes de contribuir para a unicidade do *fingerprint*, mas que podem ser usados para compor outros atributos, como foi o caso do Canvas do método proposto, que se mostrou o atributo mais relevante.

Outra observação a ser feita é sobre o atributo *Media Devices*. Na entropia de Shannon o atributo alcançou relevância acima da média (2,94), já no InfoGain,

Tabela 6.3: Grau de relevância dos atributos segundo o InfoGain.

Atributo	**Relevância**
Canvas	6,6272
WebGL Renderer	5,6578
Khademi	5,5746
User Agent	5,0964
Laperdrix	4,9924
Waves	3,41
WebGL Vendor	2,7748
Platform	2,6964
Media Devices	2,0354
Resolution	1,6878
Hardware Concurrency	1,4726
OfflineAudioContext	0,6072

ficou abaixo da média. Isso aconteceu porque o InfoGain considerou todos dados obtidos do *Media Devices*, mesmo quando os dados eram nulos. Dessa maneira, de acordo com o InfoGain, o *Media Devices* é um atributos que isoladamente não faz sentido, pois pode ser melhor aproveitado na composição de atributos.

6.3 Canvas Composto vs Khademi vs Laperdrix

Esta Seção compara os dois métodos de Canvas *fingerprinting* encontrados na literatura que obtiveram as maiores acurácias com o Canvas *fingerprinting* encontrado no método proposto nesta pesquisa. Para realizar a comparação, considerou-se o grau de entropia. Essa comparação é realizada com o objetivo de avaliar se um Canvas composto por outros atributos resulta em um Canvas com maior unicidade do que aqueles não compostos por outros atributos.

A Tabela 6.4 mostra a entropia do método de Canvas desenvolvido nesta pesquisa e os Canvas encontra nos trabalhos relacionados [1] e [2].

Tabela 6.4: Comparação dos Canvas.

Autor	**Entropia do Canvas**
Pesquisa	6,6272
Khademi et. all (2015)	5,5745
Laperdrix et. all (2016)	5,0062

Comparando em termos numéricos os três Canvas: Canvas composto, Kha-

demi et al. e Laperdrix et al., pode-se notar que o primeiro possui maior relevância do que os dois últimos, tanto na entropia de Shannon quanto no InfoGain. No entanto, é necessário entender quais elementos causam essa diferença.

O Canvas apresentado em Khademi et al. [1] é composto por duas formas geométricas, de cores distintas e com pantograma repetido diversas vezes em diferentes cores, com o objetivo de explorar mais capacidade de processamento da GPU. Por outro lado, o Canvas apresentado em Laperdrix et al. [2] não explora cores, mas sim o uso de emojis, fontes falsas e, assim como [1], pantograma, com o mesmo objetivo de obter mais características relativas da GPU.

Desta forma, pode-se ver uma grande diferença entre esses dois métodos de Canvas apresentados e o proposto nesta dissertação: a dinamicidade. Os dois primeiros Canvas são visualmente estáticos e depende apenas da GPU, não ocorre grandes variações. Por outro lado, o Canvas proposto varia de acordo com o hardware dos dispositivos, pois os elementos desenhados no Canvas estão estritamente relacionados a outras características extraídas dos dispositivos. Dessa forma, a diferença no Canvas não depende apenas da capacidade da GPU renderizar a mesma imagem e daí obter a organização dos pixels, mas depende de como os outros atributos do dispositivo impactam na formação de pixels no Canvas, que por sua vez impacta em como a GPU vai renderizar a imagem. Assim, é possível notar que o Canvas proposto depende mais do que um elemento de hardware, mas sim de 5 atributos (*OfflineAudioContext*, ondas periódicas, número lógicos de núcleos, *WebGL vendor* e *WebGL renderer*. Essa diversidade colabora para um maior grau de aleatoriedade, que por sua vez colabora com a unicidade dos *fingerprints* gerados pelo método proposto nesta dissertação.

6.4 Comparação entre Métodos

Está Seção apresenta a comparação entre o método proposto e aqueles encontrados nos trabalhos relacionados, com o objetivo de averiguar a acurácia deste método em relação aos outros. No entanto, para que a comparação entre os métodos seja justa, é necessário utilizar a entropia ponderada [2], que é calculada de acordo com a equação 6.1:

$$\eta(X) = \frac{H(X)}{\log_2(n)} \tag{6.1}$$

Onde *H(X)* é a entropia de Shannon e n é o número de *fingerprints* contidos na base de dados, duplicados ou não. Dessa maneira, é possível comparar dois ou mais métodos de *Web Fingerprinting* sem a necessidade de se preocupar com o número de *fingerprints* obtidos pelos métodos, pois essa equação considera a distribuição das probabilidades.

Vale ressaltar que, embora a entropia ponderada pode ser aplicada para fazer comparação entre bases de dados de tamanhos distintos, a base de dados obtida durante o experimentos é muito menor do que aquelas encontradas nos trabalhos relacionados. Isso significa que, conforme o conjunto de chaves for aumentando, a precisão do método pode diminuir em relação os trabalhos relacionados.

Aplicando a equação 6.1 nos trabalhos relacionados [1, 2, 9, 11, 13, 15] e no método desenvolvido, tem-se a entropia ponderada para cada um dos métodos descrita na Tabela 6.5. Vale ressaltar que o trabalho de [5] não foi considerado pelo foco ser identificar classes de dispositivos, não identificá-los de forma única.

Tabela 6.5: Entropia ponderada dos métodos de *Web Fingerprinting*.

Autor	**# Atributos**	**Entropia ponderada**
Khademi et. all (2015)	74	0,9723
Método Proposto	10	0,9707
Nakibly et. all (2015)	2	0,7319
Mowery et. all (2012)	2	0,6988
Ximenes et. all (2016)	3 (Canvas)	0,4922
Englehardt et. all (2016)	1	0,3809
Laperdrix et. all (2016)	18 (Canvas)	0,491

Pode-se notar na Tabela 6.5 que o método mais eficaz é o de [1], com entropia ponderada acima de 95%. No entanto, vale ressaltar que um total de 74 atributos foram usados, que por sua vez são oriundos do JavaScript (fontes e objetos), Flash e Canvas. O segundo método de *Web Fingerprinting* que se destaca é aquele proposto nesta dissertação, que utiliza 10 atributos, em sua maioria dependentes das características de hardware. Embora em segundo lugar, pode-se perceber que o método proposto emprega sete vezes menos atributos que o método apresentado por [1] e que a diferença na entropia ponderada entre os dois está abaixo de 0,1 grau (0,0016).

O terceiro método que mais se destaca é o de [11], que faz o *fingerprint* com base na frequência de *clock* e no *clock skew* da GPU, obtidos por efeitos colaterais no Canvas. Dessa forma, foi considerado que dois atributos foram utilizados. O método de *Canvas Fingerprinting* proposto por [9] fica em quarto. É considerado o uso de dois atributos porque o Canvas é usado de duas formas: Uma com imagens em 2D e a outra em 3D. Vale ressaltar que o método proposto pelos autores foi o precursor em métodos de *Canvas Fingerprinting*. O quinto método mais eficaz é apresentado por [13], que utiliza três Canvas para gerar o *fingerprint*, cada um com entropias distintas: L, H e M, que representam baixa, média e alta entropia, respectivamente.

O sexto método listado na Tabela 6.5 é de *AudioContext Fingerprinting*, es-

tudado por [15]. Dessa forma, a entropia pondera foi calculada apenas para esse atributo. Comparando o método de *OfflineAudioContext* desenvolvido nesta pesquisa com aquele encontrado em [15], pode-se observar que o método proposto nesta pesquisa superou aquele apresentado por [15], pois o dessa pesquisa alcançou a entropia ponderada de grau 0,4749.

A entropia ponderada calculada para [2] considera apenas o Canvas utilizado no método dos autores[3]. Fazendo a comparação do Canvas apresentado em [2] com aquele proposto nesta pesquisa, pode-se notar que o Canvas apresentado por este trabalho apresenta grau de entropia ponderada superior (0,9230).

O método proposto nesta dissertação é tão eficiente quanto aqueles propostos na literatura, tendo como diferencial a baixa quantidade de atributos e a entropia similar ao método proposto por [1], que mais se destacou na comparação. Dessa maneira, pode-se notar também que ao utilizar atributos relacionados às características de hardware, o método de *Web Fingerprinting* torna-se menos dependente do emprego de muitos atributos para identificar dispositivos. Essa observação se aplica mesmo que parte dos atributos possuam baixa entropia quando isolados. Nesses casos, a composição de atributos, isto é, usar atributos para compor outros, torna-se uma das estratégias que podem ser empregadas para projetar métodos de *Web Fingerprinting*.

Mesmo que a proposta apresentada neste trabalho seja tão eficiente quando aquelas encontradas na literatura, vale ressaltar que o método proposto possui como ponto fraco máquinas clonadas, isto é, dispositivos que possuem o mesmo conjunto de *hardware* e de *software*.

Para melhor exemplificar a situação acima, dos 14 dispositivos com chaves duplicadas, três deles eram um Moto G4 que estavam executando o navegador Google Chrome. Esse comportamento é explicado pelo fato do método não forçar o dispositivo, pois isso pode causar efeitos colaterais, como por exemplo, travamento da página ou até mesmo do navegador.

[3]Os autores não informaram a entropia de sua base de dados, dessa forma não foi possível calcular a entropia ponderada para todos os atributos

Capítulo 7

Avaliando as contramedidas

Este Capítulo discorre sobre as contramedidas encontradas na literatura e em lojas de extensões para navegadores[1]. Além da descrição, este Capítulo também apresenta os efeitos que as contramedidas causam no método de *Web Fingerprinting* desenvolvido neste trabalho.

7.1 Contramedidas

Extensões e procedimentos cujo objetivo seja dificultar ou frustar os mecanismos de *fingerprinting* são vistos como contramedidas ao *fingerprinting*. Dentre eles, pode-se destacar certas extensões para navegadores *Web*: Adblock Plus, Blur, Disconnect, Ghostery, Privacy Badger, Privacy Protector Plus, uBlock Origin, Canvas Fingerprinting Blocker, No-Script Suite Lite e User-Agent Switcher, por exemplo.

As contramedidas, de acordo com [29] e [8], trabalham de duas formas. A primeira é pelo uso de heurísticas, onde um conjunto de domínios suspeitos de rastrear usuários são registrados em uma base de dados centralizada. Nesse modelo, quando a extensão de contramedida percebe que o domínio acessado pelo usuário é suspeito de violar sua privacidade, um aviso de alerta é dado ao usuário ou até mesmo a página não é carregada. O problema desse modelo é que a contramedida precisa de heurísticas para detectar novos rastreadores e manter a base atualizada. No entanto, nem sempre a heurística pode funcionar na medida em que novos métodos possam surgir [29].

O segundo modelo utiliza filtros (*black* ou *white list*) para frustar os mecanismos de *fingerprinting*. Tal modelo depende da interação do usuário, onde o mesmo configura a ferramenta para bloquear a execução de *scripts* oriundos de sites considerados não confiáveis. A diferença desse modelo para o anterior, é que

[1]Para o Chrome, Firefox e Opera, devido a alta disponibilidade de extensões.

esses filtros são definidos manualmente, de acordo com as preferências do usuário [8].

Os modelos podem ser combinados para criar extensões capazes de combater os métodos de *Web Fingerprinting*, por meio da mutação de agentes (fazer um sistema A se passar por um sistema B) e até mesmo por adições de ruídos nos dados extraídos. Por exemplo, uma extensão com essa característica é capaz de fazer com que um Firefox rodando em um PC com Windows se passe por um Safari rodando em um iPhone. Outra aplicação para esse modelo é quando ruídos são acrescentado nos dados extraídos. Por exemplo, *pixels* modificados adicionados em um HTML5 Canvas no momento em que seu conteúdo é lido, como apresentado em [30].

Algumas ferramentas que utilizam tal modelo fazem uso de randomização, em que a cada intervalo de tempo, número de requisições ou outros fatores (definidos ou não pelo usuário) modifica o agente do navegador ou acrescentam ruídos nos dados extraídos, fazendo com que o domínio enxergue sempre um dispositivo diferente ou até mesmo desconhecido, dificultando a identificação acurada do usuário que acessa o domínio. Um exemplo disso é a pesquisa desenvolvida em [31].

Das contramedidas citadas no início dessa Seção, aquelas que são baseadas em filtros definidos pelos usuários são as seguintes: Adblock Plus[2], Canvas Fingerprinting Blocker, Ublock Origin, No Script Suite Lite, User-Agent Switcher e Stop Fingerprinting. As contramedidas que são baseadas em heurísticas são as seguintes: Blur, Disconnect, Ghostery, Privacy Badger e Privacy Protector Plus. Vale ressaltar que o Privacy Badger também pode ser configurado pelo usuário para indicar se um domínio é rastreador ou não. Vale ressaltar que essa classificação decorreu do experimento descrito mais adiante.

Além de serem contramedidas ao *Web fingerprinting*, elas possuem outra característica em comum: são populares em lojas de extensões e fazem-se presentes nos navegadores Chrome, Firefox e Opera na versão Desktop (com exceção do Canvas Fingerprinting Blocker e Disconect). No caso do Privacy Badger, é uma extensão que foi desenvolvida como resultado do projeto Panopticlick [6]. A Tabela 7.1 descreve essas contramedidas e suas características.

7.2 Avaliação das Contramedidas

Para avaliar o impacto da ação das contramedidas no método de Fingerprinting proposto, alguns testes foram executados nos quatro navegadores mais populares

[2]O Adblock Plus depende de filtro para o caso de bloquear a execução de *scripts*, não de propagandas.

Tabela 7.1: Breve descrição das contramedidas.

Extensão	**Características**
Adblock Plus	Bloqueia os pedidos HTTP de acordo com o endereço de origem do *script*, podendo bloquear IFrames, JavaScript e Flash. Também permite que o usuário crie lista de *sites* não confiáveis para que os elementos dinâmicos dos mesmos sejam bloqueados.
Blur	Possui uma lista interna de domínios oriundos de companhias conhecidas por rastrear usuários. A lista interna é atualizada de acordo com os *feedbacks* dos usuários, permitindo que novos rastreadores sejam bloqueados. O *script* é capaz de bloquear rastreadores estáticos (*cookies*) e dinâmicos (*fingerprinting*).
Ghostery	Bloqueia *scripts* que coletam dados sobre os hábitos de navegação dos usuários, de acordo com o endereço do domínio. Além disso, a biblioteca de *scripts* conhecidos é atualizada de forma automática, a medida em que várias páginas *Web* são exploradas em busca por novos rastreadores.
Privacy badger	Baseado do Adblock Plus, mas apenas bloqueia propagandas que contém códigos rastreadores.
uBlock Origin	Depende de listas compiladas por comunidades, cujo conteúdo são domínios conhecidos por rastrear usuários. Além disso, a extensão permite que os usuários importem arquivos com domínios para serem bloqueados e que criem listas pessoais de sites não confiáveis.
NoScript	Bloqueia JavaScript, Java, Fash, Silverlght e outros recursos dinâmicos presentes nos navegadores, com o objetivo de evitar que o usuário seja rastreado. O usuário pode criar lista de sites confiáveis e não confiaveis, em que a execução de conteúdos dinâmicos são autorizadas ou não.
Disconnect	Utiliza *Web Crawlers* para buscar e detectar *scripts* que rastreiam usuários. Quando descobertos, o domínio de origem é registrado, então todas as requisições desse domínio são bloqueadas no navegador em que a extensão encontra-se instalada.
Privacy Protector Plus	Permite que o usuário crie listas de *sites* confiáveis que possuem permissão para executar *scripts*. Além disso, possui uma lista fixa embutida que se atualiza sempre que um *script* rastreador é detectado, com o objetivo de bloqueá-lo.
User-Agent Switcher	Permite que o usuário forje o *user-agent* do navegador que está utilizando. A ideia se baseia no fato de que a mudança eventual do *user-agent* poderá dificultar reidentificar um dispositivo.
Stop Fingerprinting	Desativa e/ou modifica algumas APIs do JavaScript, por exemplo, *screen*, *MediaDevices*, *navigator* e *plugins* do objeto *window*. Como a desativação ou modificação pode afetar de forma negativa o funcionamento de certas páginas, o *plugin* permite que o usuário crie listas de *sites* confiáveis e não confiáveis.
Canvas Fingerprinting Blocker	Bloqueia a criação do elemento Canvas ou a recuperação de seu conteúdo. Em certos *plugins* que bloqueiam o Canvas, ruídos são adicionados na imagem gerada, com o objetivo de distorcer o *fingerprint* e tornar mais difícil a reidentificação de usuários.

em *Desktops*[3] até a data de escrita deste trabalho e que possuem diversas extensões de contramedidas em suas lojas: Chrome, Firefox e Opera. O motivo do *Desktop* ter sido escolhido é que dispositivos móveis não suportam extensões [7].

Antes do experimento para testar o comportamento do método de *Web Fingerprinting* em relação as extensões de contramedidas, foi necessária a instalação de cada uma das extensões apresentadas acima, dentre outras que fizeram parte do experimento, que embora não foram citadas, são equivalentes àquelas apresentadas anteriormente.

7.2.1 Experimento

Após a instalação, o experimento ocorreu da seguinte maneira: Para cada navegador (Chrome, Firefox e Opera) e para cada uma das extensões instalada nos navegadores, a página *Web* localizada em servidor local (127.0.0.1) foi requisitada duas vezes, a primeira com a extensão desativada e a segunda com a extensão ativada. Ao final de cada requisição, a saída do método foi observada. O critério para determinar se a contramedida afetou o comportamento do método foi o seguinte: Se para as duas requisições a saída foi igual, então a extensão não influenciou no método. Por outro lado, se a saída da segunda requisição foi diferente da primeira ou se o método não foi executado, então a extensão influenciou o método. Ao todo, foram 34 iterações em um computador com SO Ubuntu 16.04, com processador Intel 1.7 de quarta geração, 4GB de RAM e placa gráfica Intel HD graphics.

A Tabela 7.2 lista as extensões agrupadas por navegador, bem como o resultado do experimento realizado em cada uma delas.

No primeiro grupo (Chrome, Firefox e Opera) é possível notar que as extensões que foram capazes de comprometer o funcionamento do método desenvolvido neste trabalho são o Adblock Plus, No-Script Suite Lite e UBlock Origin. O que essas extensões têm em comum é que em todas elas foi configurado um filtro que o domínio 127.0.0.1 deve ser bloqueado. Por outro lado, as outras extensões que não funcionaram, provavelmente consideram apenas se o domínio em que o *script* rodou era confiável ou não, talvez com base em uma lista de domínios confiáveis ou em *scripts* já conhecidos. No caso do User-Agent Switcher, a extensão não bloqueou o método, mas a mudança de User-Agent gerou um novo *fingerprint*.

No segundo grupo (Chrome e Firefox), o Canvas Fingerprinting Blocker funcionou, enquanto o Disconnect não. A primeira extensão funcionou porque a mesma requer a interessão do usuário, informando que a página usa o elemento canvas e posteriormente pergunta se o elemento canvas deve ser bloqueado ou

[3]De acordo com a página http://gs.statcounter.com/browser-market-share/desktop/worldwide

Tabela 7.2: Experimento das contramedidas. F: Funcionou. NF: Não funcionou.

Navegador	Extensão	Resultado
Chrome, Firefox e Opera	Adblock Plus	F
	Blur	NF
	Ghostery	NF
	No-Script Suite Lite	F
	Privacy Badger	NF
	Privacy Protector Plus	NF
	UblockOrigin	F
	User-Agent Switcher	F
Chrome e Firefox	Canvas Fingerprinting Blocker	F
	Disconnect	NF
Chrome e Opera	ScriptSafe	F
Firefox	Stop Fingerprinting	NF
Chrome	Better, Faster, Private Browsing	NF
	Random User-Agent	F

não. Já a segunda extensão não requer a interação do usuário, provavelmente dependendo a análise do domínio ou de alguma base de dados de domínios suspeitos.

No terceiro e quarto grupo (Chrome e Opera) e (Firefox) pode-se notar que, mais uma vez, extensões que utilizam filtro definido pelo usuário mostram que são capazes de comprometer o método proposto neste trabalho, enquanto as que utilizam-se de alguma análise do domínio ou de base de dados, mostram-se ineficazes.

No quinto e último grupo (Chrome), a extensão Better, Faster, Private Browsing não compromete o método de *fingerprinting*, pois deve ser baseada em alguma base de dados. Já a Random User-Agent prejudica o método, devido o campo User-Agent ser usado no momento de gerar um *fingerprint*.

7.3 Considerações do Capítulo

As contramedidas testadas, embora funcionem no ambiente aberto da *Web*, pode-se notar que ainda assim possuem suas limitações. Por exemplo, em um ambiente local, as ferramentas que deveriam detectar *scripts* de *fingerpriting* não foram capazes de fazer a detecção, o que pode abrir portas para novos domínios conseguirem rastrear usuários no curto a médio prazo. As contramedidas que dependem de uma lista difundida entre usuários e configuradas localmente são eficazes, mas também possuem suas limitações, pois novos domínios podem não ser registra-

dos por determinado tempo. Além disso, configurar listas manualmente pode ser uma tarefa árdua para usuários que não se interessam pelo assunto.

Capítulo 8

Considerações Finais

Técnicas de *Web Fingerprinting* podem ser implementadas de diversas maneiras, com o objetivo de identificar usuários e, em certos casos, identificar a classe ou o tipo dos dispositivos. Para alcançar tais objetivos, é necessário selecionar os atributos mais relevantes e que, ao mesmo tempo, sejam estáveis, isto é, que se mantenham inalterados ao longo o tempo, para facilitar a re-identificação dos usuários.

Tendo em vista a importância dos atributos, essa dissertação apresentou uma proposta de método de *Web Fingerprinting* que utiliza atributos em sua maioria relacionados a características de hardware, para identificar os dispositivos e seus respectivos usuários. O método foi comparado com aqueles encontrados na literatura, utilizando-se como métrica de avaliação a entropia de Shannon (padrão e ponderada). Também investigou três pontos principais: Canvas, composição de atributos e a Web Audio API, para obter características de áudio. A Web Audio API foi investigada em dois pontos: AudioContext e OfflineAudioContext, para processar áudio com a finalidade de obter os sinais resultantes e usá-los como características relevantes para o método de *Web Fingerprinting* proposto. Além disso, notou-se também que a utilização de atributos mais fracos (Waves, MediaDevices, etc.) para compor os atributos mais fortes (Canvas, por exemplo) contribui com o método no sentido de enriquecer as características obtidas dos dispositivos. Dessa maneira, o método proposto, assim com os trabalhos encontrados na literatura, demostra o quão vulnerável os usuários estão na Web no que diz respeito a privacidade. Embora haja esse risco, é possível utilizar *Web Fingerprinting* para fins benéficos. Aplicações Web que necessitam de autenticação, podem utilizar métodos de Web Fingerprinting em conjunto com outras formas de autenticação e, dessa maneira, incrementar a segurança.

No quesito de avaliação, foi demonstrado que o método proposto é tão eficaz quanto os trabalhos relacionados, mas com a diferença de utilizar 10 atributos, onde a maioria são relacionados a caraterísticas de hardware.

8.1 Dificuldades Encontradas

Diversas dificuldades foram enfrentadas no decorrer da pesquisa, algumas das principais foram as seguintes.

A obtenção de ondas periódicas com o AudioContext foi uma tarefa complexa que requeria o *timing* de funções para iniciar, obter e finalizar a emissão do sinal. Essas operações resultavam em *samples* instáveis. Mesmo utilizando um tratador de eventos para obter os sinais do AudioContext, o conjunto de frequências era instável, impactando negativamente no *fingerprint* resultante.

A Web Audio API fornece diversos nós de áudio e permite montar grafos de áudio de diversas topologia. Foi desafiador escolher os nós e suas configurações para então implementar um teste pequeno e averiguar se os *fingerprints* obtidos eram estáveis. Com diversas interfaces disponíveis para uso, várias combinações podem ser feitas. Este trabalho apresenta apenas uma delas.

Os testes para averiguar se as frequências obtidas eram estáveis ou não, demandou trabalho e foi um tanto complexa, pois foi necessário repetir o experimento diversas vezes em vários navegadores e dispositivos. Também foi necessário verificar se os valores de cada atributo eram retornados, pois se não fossem seria necessário descobrir a causa e então verificar tal limitação iria inviabilizar o método proposto nesta pesquisa.

Com o OfflineAudioContext e o Canvas implementados, um experimento foi realizado considerando apenas ambos os atributos, para investigar se somente o Canvas e o OfflineAudioContext seriam capazes de gerar *fingerprints* para identificar, de forma única, os usuários. Como resultado, foi constatado que não, pois mais atributos seriam necessários.

Dessa maneira, foi necessário encontrar mais atributos que pudessem fornecer características relacionadas ao hardware, mas que não fossem tão fáceis de manipular por extensões de contramedida.

8.2 Contribuições Alcançadas

Este trabalho contribuiu com a seleção de atributos relevantes relacionados a caraterísticas de hardware que podem ser usados em métodos de *Web Fingerprinting*.

O método proposto demostra que, mesmo com poucos atributos (em comparação com os trabalhos relacionados), é possível obter um método de *Web Fingerprinting* tão eficaz quanto aqueles que utilizam vários atributos, contanto que sejam relevantes, como mostrado com o uso da entropia de Shannon e com o *rank* do InfoGain.

Além disso, como um dos resultados desse trabalho, o artigo “Um método de

identificação de navegadores Web baseado na Web AudioAPI" foi aceito no XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg2016).

8.3 Trabalhos Futuros

Embora possa ser utilizado para identificar dispositivos e consequentemente seus donos, fica evidente que certos atributos (Media Devices, por exemplo) falham na maioria dos positivos. Dessa forma, é necessário encontrar atributos relevantes, mas que também sejam compatíveis com navegadores ou que não dependam da permissão do usuário para serem utilizados.

Embora o OfflineAudioContext tenha se mostrado a interface mais promissora da Web Audio API, quando comparado com o AudioContext, é necessário investigar o motivo pelo qual o AudioContext se comporta de forma instável. Tendo em vista que o sinal é renderizado no *hardware*, tal atributo pode ser relevante se gerar chaves estáveis ao longo do tempo.

É necessário investigar se o modo (isto é, as configurações) dos AudioNodes utilizados para montar o grafo de áudio foi a mais eficiente, tendo em vista que a Web Audio API compreende vários AudioNodes e diversas possibilidades de configuração. Além disso, extrair características dos dispositivos é a única forma de fazer o *fingerprinting*? E se a Web Audio API fosse usada de forma a também fazer o *fingerprinting* do ambiente por meio de amostras de som?

Referências Bibliográficas

[1] A. F. Khademi, M. Zulkernine, and K. Weldemariam, "An Empirical Evaluation of Web-Based Fingerprinting," *IEEE Software*, vol. 32, no. 4, pp. 46–52, 2015.

[2] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints," in *IEEE Symposium on Security and Privacy (SP)*, pp. 878–894, IEEE, 2016.

[3] N. Nikiforakis and G. Acar, "Browse at your own risk," *IEEE Spectrum*, vol. 51, no. 8, pp. 30–35, 2014.

[4] R. Upathilake, Y. Li, and A. Matrawy, "A classification of web browser fingerprinting techniques," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, IEEE, 2015.

[5] E. Bursztein, A. Malyshev, T. Pietraszek, and K. Thomas, "Picasso: Lightweight Device Class Fingerprinting for Web Clients," in *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pp. 93–102, ACM, 2016.

[6] P. Eckersley, "How Unique is Your Web Browser?," in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, (Berlin, Germany), pp. 1–18, Springer-Verlag, 2010.

[7] T. Hupperich, D. Maiorca, M. Kührer, T. Holz, and G. Giacinto, "On the robustness of mobile device fingerprinting: Can mobile users escape modern Web-tracking mechanisms?," in *Proceedings of the 31st Annual Computer Security Applications Conference*, pp. 191–200, ACM, 2015.

[8] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, "Fingerprinting Information in JavaScript Implementations," in *Proceedings of W2SP 2011* (H. Wang, ed.), pp. 1–11, IEEE Computer Society, 2011.

[9] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," *Proceedings of W2SP*, pp. 1–12, 2012.

[10] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, E. Weippl, and F. Wien, "Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting," in *Web 2.0 Workshop on Security and Privacy (W2SP)*, vol. 5, 2013.

[11] G. Nakibly, G. Shelef, and S. Yudilevich, "Hardware Fingerprinting Using HTML5," *CoRR*, vol. abs/1503.0, 2015.

[12] T. Unger, M. Mulazzani, D. Fruhwirt, M. Huber, S. Schrittwieser, and E. Weippl, "SHPF: Enhancing HTTP(S) session security with browser fingerprinting," in *2013 International Conference on Availability, Reliability and Security*, pp. 255–261, IEEE, 2013.

[13] P. Ximenes, M. Correia, P. Mello, F. Carvalho, M. Franklin, and R. Andrade, "TARP Fingerprinting: Um Mecanismo de Browser Fingerprinting Baseado em HTML5 Resistente a Contramedidas," in *Anais do XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pp. 100–113, SBC, 2016.

[14] Optanon, "The Cookie Law Explained." https://www.cookielaw.org/the-cookie-law/. Acessado em 11/04/2017.

[15] S. Englehardt and A. Narayanan, "Online Tracking: A 1-million-site Measurement and Analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1388–1401, ACM, 2016.

[16] F. Alaca and P. C. V. Oorschot, "Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 289–301, ACM, 2016.

[17] A. Saraiva, E. Feitosa, P. Elleres, and G. Carneiro, "Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas," in *Livro de Minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg2014*, pp. 49–98, SBC, 2014.

[18] W3C, "Fingerprinting Guidance for Web Specification Authors (Draft)." https://www.w3.org/TR/fingerprinting-guidance/, 2015. Acessado em 01/01/2016.

[19] K. Takasu, T. Saito, T. Yamada, and T. Ishikawa, "A Survey of Hardware Features in Modern Browsers: 2015 Edition," in *2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 520–524, IEEE, 2015.

[20] A. Faiz Khademi, *Browser Fingerprinting: Analysis, Detection, and Prevention at Runtime*. Master's thesis, School of Computing, 2014. Queen's University.

[21] W3schools, "HTML5 Canvas." http://www.w3schools.com/html/html5/canvas.asp. Acessado em 03/05/2016.

[22] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The Web Never Forgets: Persistent Tracking Mechanisms in the Wild," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 674–689, ACM, 2014.

[23] MDN, "Web audio: conceitos e uso." https://developer.mozilla.org/pt-BR/docs/Web/API/API_Web_Audio. Acessado em 01/01/2016.

[24] T. M. Cover and J. A. Thomas, "Elements of information theory 2nd edition (wiley series in telecommunications and signal processing)." Acessado em 2006.

[25] Khronos, "OpenGL ES for the Web." https://www.khronos.org/webgl/. Acessado em 03/05/2016.

[26] C. E. Shannon, "Prediction and entropy of printed english," *Bell Labs Technical Journal*, vol. 30, no. 1, pp. 50–64, 1951.

[27] W3C, "Web Audio API." https://www.w3.org/TR/2015/WD-webaudio-20151208/, 2015. Acessado em 15/08/2017.

[28] Y. Cao, S. Li, and E. Wijmans, "browser fingerprinting via os and hardware level features," in *Proceedings of Network & Distributed System Security Symposium (NDSS)*, 2017.

[29] C. F. Torres, H. Jonker, and S. Mauw, "FP-block: Usable web privacy by controlling browser fingerprinting," in *European Symposium on Research in Computer Security*, pp. 3–19, Springer, 2015.

[30] P. Baumann, S. Katzenbeisser, M. Stopczynski, and E. Tews, "Disguised Chromium Browser: Robust Browser, Flash and Canvas Fingerprinting Protection," in *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, pp. 37–46, ACM, 2016.

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

Apêndice A

Fingerprints Gerados

Este apêndice apresenta duas tabelas com os *fingerprints* gerados durante os experimentos desta dissertação. A primeira (Tabela A.1) apresenta os resultados (*fingerprints*) do experimento realizado com o *OfflineAudioContext* explicado no Capítulo 4. A segunda (Tabela A.2) apresenta os resultados (*fingerprints*) do experimento final realizado com o método proposto explicado no Capítulo 6.

Na Tabela A.2 tem-se uma amostra de algumas chaves que foram geradas. As colunas presentes na Tabela A.1 também estão incluídas nesta.

Como é possível observar na Tabela A.2, 10 dispositivos (linhas 1 à 10) foram identificados unicamente, um número maior quando comparado com a Tabela A.1. Isso demonstra que a *Web Audio API*, Canvas e a composição de atributos contribuem para gerar identificadores únicos para os dispositivos.

Ainda na Tabela A.2 é possível observar que dispositivos, à primeira vista idênticos, na verdade podem ser diferenciados um do outro, como registrado nas linhas 6 e 7, onde tem-se dois *smartphones* ou *tablets* distintos que fazem uso do SO Android 7 e do navegador Chrome na versão 63.0.

Embora o método seja capaz de identificar dispositivos de forma única, com precisão de 90,30%, vale ressaltar que mesmo assim o método possui suas limitações, como pode ser observado nas linhas 11 à 20. É possível perceber que dispositivos semelhantes ainda podem compartilhar a mesma chave, o que pode ser causado por dispositivos muito semelhantes ou porque as características extraídas não foram o suficiente para distingui-los.

Tabela A.1: *OfflineAudioContext fingerprints* gerados e seus atributos.

	Chave	Navegador	Versão (Navegador)	SO	Versão (SO)	Dispositivo
1	12916ca56 ...	Chrome	58.0.3029.110	OS X	10.12.5	Destkop/Notebook
2		UR	55.1.2883.7			
3	1bd2be502 ...	Chrome	58.0.3029.83	Android	5	Smartphone/Tablet
4	412e24302 ...	Firefox	47.0	Windows	7	Destkop/Notebook
5	559f0ecae ...	Edge	15.15063	Windows	10	Destkop/Notebook
6	64e7af5ca ...	Chrome	58.0.3029.96 - 51.0.2704.79 - 55.0.2883.87 - 59.0.3071.86 - 56.0.2924.87 - 59.0.3071.104 - 58.0.3029.110 - 54.0.2840.90 - 57.0.2987.133 - 59.0.3071.109 - 58.0.3029.81 - 59.0.3071.86 - 51.0.2704.106	Linux	–	Destkop/Notebook
7			58.0.3029.110 - 58.5.3029.81	Windows	10	
8		Opera	46.0.2597.26	Linux	–	
9			45.0.2552.898	Windows	10	
10			45.0.2552.898	Windows	7	
11		Chromium	53.0.2785.143	Linux	–	
12	654fe84fe ...	Chrome	59.0.3071.86	Linux	–	Destkop/Notebook
13			58.0.3029.110 - 59.0.3071.109	Windows	7	
14			58.0.3029.110 - 59.0.3071.109 - 59.0.3071.104	Windows	10	
15			58.0.3029.110	Windows	8	
16		Opera	45.0.2552.898	Windows	10	
17		Chromium	51.0.2683.0	Windows	10	
18	6dc773aa2 ...	Opera	47.0.2628.0	OS X	10.12.4	Destkop/Notebook
19		Chrome	58.0.3029.110	OS X	10.12.4	
20	73e92b923 ...	Safari	10.1.1	OS X	10.12.5	Destkop/Notebook
21			10.1	OS X	10.12.4	Destkop/Notebook
22	749f38fac ...	Chrome	58.0.3029.83 - 51.0.2704.106 - 59.0.3071.92	Android	7.0	Smartphone/Tablet
23			58.0.3029.83 - 43.0.2357.93;	Android	6.0.1	
24			58.0.3029.83 - 43.0.2357.93	Android	5.1.1	
25			58.0.3029.83	Android	4.2.2	
26			58.0.3029.83	Android	7.1.2	
27			56.0.2924.87	Android	5.0	
28		Samsung Broswer	5.4	Android	5.0.2	
29	78a323009 ...	Firefox	53.0	Android	6.0.1	Smartphone/Tablet
30			53.0	Android	5.0.2	
31			54.0	Android	5.1.1	
32	91f4b87d3 ...	Firefox	54.0	Linux/Ubuntu	–	Destkop/Notebook
33			55.0	Linux	–	
34			53.0 e 54.0	Windows	10	
35			54.0	Windows	7	
36			54.0	OS X	10.12	
37			54.0	OS X	10.11	
38	c649ae295 ...	Chrome	58.0.3029.83	Android	6.0.1	Smartphone/Tablet
39			58.0.3029.83	Android	5.1.1	
40			58.0.3029.83	Android	4.4.2	
41		Opera	42.7.2246.114996	Android	6.0.1	
42	d1998f0cc ...	Chrome	59.0.3071.104	Linux	–	Destkop/Notebook
43	e5f4d3534 ...	Safari	10.1.1	OS X	10.12.5	Destkop/Notebook
44	eed33e7c6 ...	Firefox	44.0	Windows	7	Destkop/Notebook
45	038e78e78 ...	Internet Explorer	11	Windows	10	Desktop/Notebook
46			11	Windows	8.1	Smartphone/Tablet
47		Safari	9	iOS	9.3	Smartphone/Tablet
48			10	iOS	10.3.2	
49			11	iOS	11.0	
50		Chrome	59.0.3071.102	iOS	10.3	Smartphone/Tablet
51			59.0.3071.109	Linux	–	Desktop/Notebook

Tabela A.2: Amostra dos *fingerprints* gerados com o método proposto.

	Chave	Navegador	Versão (Navegador)	SO	Versão (SO)	Dispositivo
1	d73552...	Chrome	63.0	Android	7.0	Smartphone/Tablet
2	388ca2...	Chrome	53.0	Android	5.0	Smartphone/Tablet
3	3b2a15...	Edge	41	Windows	10	Desktop/Notebook
4	f9d1ee...	Firefox	58.0	Linux	–	Desktop/Notebook
5	f4e0f9...	Firefox	57	Windows	7	Desktop/Notebook
6	ab026a...	Chrome	63.0	Android	7.0	Smartphone/Tablet
7	ce5b6c	Chrome	63.0	Android	7.0	Smartphone/Tablet
8	6c98ad...	Firefox	58.0	Linux	–	Desktop/Notebook
9	b82ede...	Edge	41	Windows	10	Desktop/Notebook
10	5dac02...	Edge	38	Windows Phone	10	Smartphone/Tablet
11	e313d8...	Chrome	63.0	Windows	10	Desktop/Notebook
12		Chrome	63.0	Windows	10	Desktop/Notebook
13	31c50f...	Chrome	64	Windows	10	Desktop/Notebook
14		Chrome	64	Windows	10	Desktop/Notebook
15	0bb35e...	Chrome	64	Android	7	Smartphone/Tablet
16		Chrome	64	Android	7	Smartphone/Tablet
17	d72177...	Chrome	63	Mac OS X	10.13	Desktop/Notebook
18		Chrome	63	Mac OS X	10.13	Desktop/Notebook
19	9dd56e...	Chrome	60	Linux	–	Desktop/Notebook
20		Chrome	63	Linux	–	Desktop/Notebook