Um método de identificação de navegadores *Web* baseado na *Web Audio* API

Jordan S. Queiroz[1], **Eduardo L. Feitosa**[1]

[1]Instituto de Computação (IComp) – Universidade Federal do Amazonas (UFAM)
CEP 69.077-000 – Manaus – AM – Brasil

{jsq,efeitosa}@icomp.ufam.edu.br

Abstract. *Web fingerprinting is a process in which the user is, with high likelihood, uniquely identified by the extracted characteristics of his / her device, generating an identifier key (fingerprint). Although it can be used for malicious purposes, Web fingerprinting can also be used for good intention: enhance usability in Web pages, enhance two-factor authentication and so on. This paper investigates the Web Audio API as a Web fingerprinting method capable of identifying the devices' class. As outcome, it is found that the method is capable of identifying the device's class, based on features like device's type, Web browser's version and rendering engine.*

Resumo. *Web fingerprinting é o processo no qual o usuário é, com alta probabilidade, identificado de forma única a partir das características extraídas de seu dispositivo, gerando uma chave identificadora (fingerprint). Embora possa ser usado para propósitos maliciosos, Web fingerprinting também pode ser usado com boas intensões: melhorar usabilidade em páginas Web, melhorar a autenticação de dois fatores e assim por diante. Esse trabalho investiga a Web Audio API como um método de Web Fingerprinting capaz de identificar a classe do dispositivo. Como resultado, foi descoberto que o método é capaz de identificar a classe do dispositivo, com base em características como tipo do dispositivo, versão e motor de renderização do navegador Web.*

1. Introdução

É fato que a *Web* é o meio mais utilizado para a realização de diferentes atividades e que todo esse conjunto de possibilidades foi alcançado devido ao rápido avanço das tecnologias *Web* (linguagens, APIs, mecanismos, entre outros) que proporcionam experiências de uso cada vez melhores. Por outro lado, essas mesmas tecnologias também podem ser empregadas para violar a privacidade dos usuários na *Web*, permitindo que os mesmos possam ser rastreados e identificados [Laperdrix et al. 2016]. Um dos meios empregados é coletar características discriminatórias presentes nos dispositivos dos usuários. Dessa forma, partindo do pressuposto que cada usuário possui seu próprio dispositivo, pode-se identificar unicamente um dispositivo e, consequentemente, seu dono. Esse processo é denominado de *Web fingerprinting*.

Existem na literatura diversos métodos de *Web fingerprinting* propostos [Bursztein et al. 2016, Laperdrix et al. 2016, Ximenes et al. 2016]. No entanto, muitos deles empregam tecnologias já consolidadas, por exemplo, os objetos *screen* e *navigator* providos pelo JavaScript. Esses objetos permitem a obtenção de informações relativas

ao navegador *Web*, *plugins* instalados no mesmo e características relativas ao dispositivo, por exemplo, dimensão de tela [Khademi et al. 2015]. Outra tecnologia utilizada é o HTML5 Canvas, que possibilita realizar desenhos e obter os *pixels* dos mesmos de forma a identificar usuários. Embora esteja perdendo o suporte, o Flash também é usado em implementações de *Web Fingerpriting*, tendo em vista que essa tecnologia é capaz de prover informações mais estritas sobre o *hardware* do dispositivo, por exemplo, fontes instaladas no sistema [Khademi et al. 2015]. Além dessas tecnologias, a análise do cabeçalho HTTP também é empregada, tendo em vista que é possível obter o *User Agent*, dentre outros dados disponibilizados pelo cabeçalho.

Neste cenário, este trabalho investiga a utilização de características e tecnologias de áudio na identificação da classe dos dispositivos. O objetivo é desenvolver um método de *Web Fingerprinting*, empregando as interfaces da *Web Audio API* – especificamente *AudioContext* e *OfflineAudioContext*, visando a identificação da classe dos dispositivos e a futura incorporação em uma solução aplicada em um cenário real. De forma sucinta, a ideia é gerar ondas periódicas, obter suas amplitudes no domínio de tempo e, então, identificar a classe dos dispositivos. A vantagem dessa abordagem é a possibilidade obter características relativas ao *hardware* do dispositivo.

No restante do artigo: a Seção 2 apresenta uma breve descrição sobre *Web Fingerprinting* e *Web Audio* API; a Seção 3 discorre sobre os trabalhos relacionados; a Seção 4 apresenta a visão geral e a implementação do método de *Web Fingerprinting* proposto; a Seção 5 apresenta os experimentos realizados para averiguar a efetividade do método proposto; a Seção 6 apresenta os resultados obtidos por meio dos experimentos; e, por fim, a Seção 7 apresenta as limitações e trabalhos futuros.

2. Web Fingerprinting e Web Audio API

Web fingerprinting é o processo no qual as características do dispositivo são extraídas para gerar uma chave identificadora (*fingerprint*) capaz de identificar, com alta probabilidade e de forma única, um usuário. Para tanto, seu funcionamento requer que o dispositivo possua capacidade de navegação por meio de navegadores *Web* (Chrome, Firefox, Edge, dentre outros) [Saraiva et al. 2014].

As características que podem ser obtidas por meio de um *Web fingerprinting* podem ser classificadas em duas categorias: *software* e *hardware*. As características relacionadas ao *software* são aquelas que dependem da configuração do navegador *Web*, tais como sistema operacional, versão e plataforma do navegador, dentre outras; e do seu ambiente de execução, como extensões, *plugins*, tipos *MIME* e fontes instaladas. O problema com essas características é a facilidade e a frequência com que são alteradas, devido a, por exemplo, atualização do navegador, instalação de novos *plugins*, novas fontes e uso de extensões de contramedida, entre outros. Em outras palavras, sua volatilidade. Já as características relacionadas ao *hardware* são aquelas que dependem dos periféricos e até mesmo dos *drivers* instalados no dispositivo. Dentre alguns exemplos, destacam-se: monitor/tela do dispositivo, placa gráfica, *drivers* e placa de áudio. Diferente das características de *software*, as características de *hardware* têm menor volatilidade.

No que diz respeito a implementação de um *Web fingerprinting*, é possível fazê-lo empregando diferentes tecnologias. As mais consolidadas incluem o uso de JavaScript, Flash, HTML5 Canvas, *cookies*, Cabeçalhos HTTP, CSS e HTML. Contudo, é possível

utilizar tecnologias ainda em estágio de desenvolvimento ou não padronizadas. Um bom exemplo é a *Web Audio API*.

De acordo com a W3C (*World Wide Web Consortium*), a *Web Audio* API é uma API em JavaScript de alto nível para o processamento e sintetização de áudio em aplicações *Web*. Ela define um conjunto de operações de áudio realizadas por meio de nós de áudio (*AudioNodes*) em contextos de áudio. Os nós são interligados entre si e formam um grafo, permitindo que o processamento do áudio siga um determinado fluxo. Dentre dos tipos de nós disponíveis, destacam-se: (I) fontes de áudio, que permitem gerar algum som ou carregar uma música; (II) filtros de efeitos de áudio, que permitem processar o áudio e adicionar efeitos; (III) destino de áudio, que geralmente são os alto-falantes do dispositivo; e (IV) nós de análise, que provêm dados sobre o áudio que está sendo processado.

Em relação à contextos de áudio, a *Web Audio API* implementa dois tipos: *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 1 ilustra um exemplo simples do funcionamento do *AudioContext* (à esquerda) e do *OfflineAudioContext* (à direita). A Tabela 1 sumariza os nós de áudio que podem ser utilizados pelo *AudioContext* e *OfflineAudioContext*.

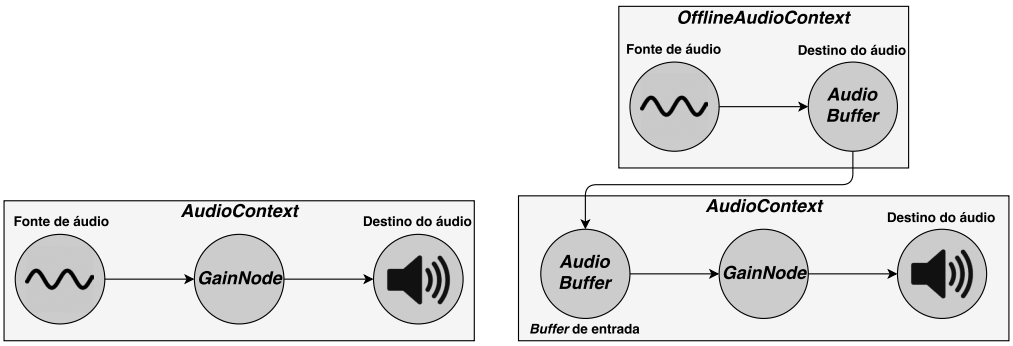


Figura 1. Exemplos de implementação do *AudioContext* (à esquerda) e do *OfflineAudioContext* (à direita).

Por também depender da placa de som do dispositivo, o *AudioContext* também está relacionado com as características de *hardware*. O processamento de áudio em diferentes máquinas e navegadores *Web* pode ser usado para obter características dos dispositivos devido a diversas combinações de *hardware* (placa de som) e *software* (*driver* do periférico) entre os mesmos, enquanto que a mesma combinação de dispositivo e navegador resultará em um *fingerprint* idêntico [Englehardt and Narayanan 2016]. Esse fenômeno pode ser atribuído à inconsistências durante o processo de fabricação dos *hardwares*, mesmo que seus modelos e número de versão sejam iguais. Essas inconsistências podem revelar, por exemplo, diferentes latências nas renderizações (no caso da

GPU) [Nakibly et al. 2015].

Tabela 1. Nós de áudio utilizados pelo *AudioContext* e *OfflineAudioContext* [MDN 2016].

Funções/Interfaces	Descrição
AudioNode	Interface que representa um módulo de processamento de áudio, por exemplo, uma fonte/destino de áudio ou módulo de processamento intermediário. O método *connect()* encontrado neste nó, permite conectar a saída de um nó *A* na entrada de um nó *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 do áudio de entrada antes que o mesmo se propague para outros nós.
AudioDestinationNode	Interface que representa o destino final de uma fonte de som em um grafo de áudio instanciado em um dado contexto, geralmente o destino é o alto-falante do dispositivo.
AnalyserNode	Interface que representa um nó capaz de prover, em tempo real, informações de análise no domínio da frequência e tempo, para o propósito de visualização de áudio. Este nó não necessita estar conectado à um nó de saída para prover os dados. Essa interface possui métodos que provêm dados sobre o áudio, dois deles são: (I) *getByteFrequencyData(Uint8Array)*: Copia os dados de frequência do sinal (no domínio da frequência) 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 *AudioBuffers*, 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 *AudioBuffer* 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) não tratados.

3. Trabalhos Relacionados

A literatura apresenta diversos trabalhos sobre *Web fingerprinting*, focando na identificação do navegador e/ou na classe do dispositivo. Alguns desses trabalhos são apresentados a seguir.

Mowery et al. [Mowery et al. 2011] implementaram e avaliaram um método de *Web Fingerprinting*, visando a identificação do navegador através da análise do desempenho e do tempo de execução do JavaScript. Para tanto, utilizaram uma combinação de 39 diferentes *benchmarks* JavaScript, bem conhecidos e bem estabelecidos, e geraram um *fingerprint* normalizado a partir de padrões de tempo de execução. Como resultado da classificação, dos 1015 casos de teste, 810 foram classificados corretamente, permitindo uma acurácia de 79.8% na identificação do navegador.

Mowery e Shacham [Mowery and Shacham 2012] propuseram um método de *Web fingerprinting* baseado na renderização de texto e gráfico em 3D no HTML5 Canvas. A ideia é que cada navegador possui um conjunto de fontes e, em certos casos, este é customizado pelo usuário. Assim, a combinação do HTML5 Canvas com fontes é capaz de extrair características discriminatórias que podem ser usadas para implementar um *Web fingerprinting*. Como resultado, o método proposto pelos autores identificou de forma única um total de 116 dispositivos do total de 294. A amostra de *fingerprint* obtida apresenta entropia de 5,73 *bits* para o Canvas nos contextos 2D e 3D. No caso do Canvas exclusivamente no contexto 2D, foi alcançada a entropia de grau 3,05 *bits*.

Mulazzani et al. [Mulazzani et al. 2013] propuseram um método de *Web Fingerprinting* que utiliza testes no motor do JavaScript para melhorar o estado-da-arte no que tange o gerenciamento de sessão HTTP, dificultando o roubo de sessão. Os autores implementaram *scripts*, baseados no padrão ECMAScript, para testar certos aspectos dos

navegadores e observaram aqueles que falharam em cada teste. Como resultado, os autores listaram os navegadores *Web* mais populares, bem como o número de testes em que os mesmos são reprovados e, dessa maneira, comprovaram que é possível identificar um navegador *Web* empregando testes de conformidade.

Khademi et al. [Khademi et al. 2015] realizaram uma análise de métodos de *Web Fingerprinting* (objetos do JavaScript, Flash, detecção de fontes por JavaScript e Canvas *fingerprinting*) para determinar o fluxo que os *scripts* de *Web Fingerprinting* seguem e, dessa forma, definir métricas para detectar tentativas de *fingerprinting* por parte de interessados. Os autores desenvolveram um *script* que emprega diversos atributos obtidos por meio dos métodos citados anteriormente. Como resultado, foram coletados 1.523 *fingerprints*, sendo que 90,41% deles são únicos. No entanto, o HTML5 Canvas colaborou com apenas 0,05% da unicidade, apresentando uma entropia de grau 3,21 bits.

Laperdrix et al. [Laperdrix et al. 2016] projetaram e avaliaram uma técnica de *Web fingerprinting* que emprega novos recursos do HTML5, com o objetivo de averiguar se contribuem significativamente na identificação de usuários. Dessa maneira, os autores exploraram a extração das características propostas por Eckersley [Eckersley 2010] (resolução de tela, *user agent*, lista de *plugins*, dentre outros) e adicionaram sete atributos (*WebGL vendor*, *WebGL renderer*, Canvas, dentre outros) que surgiram em decorrência da evolução do HTML5 e de suas APIs. Nos experimentos realizados foram coletados 118.000 *fingerprints*, sendo que 89,4% deles são únicos.

Bursztein et al. [Bursztein et al. 2016] desenvolveram e avaliaram uma técnica de *Web fingerprinting* para detectar e prevenir ataques automatizados oriundos de clientes maliciosos em lojas de aplicativos. O objetivo era distinguir clientes autênticos daqueles que são emulados ou automatizados. Diferentemente dos outros trabalhos encontrados na literatura, o método determina a classe do dispositivo, sendo composta pela tupla $\{$*navegador Web; SO*$\}$. Como resultado, o método proposto foi capaz de determinar com 100% de acurácia a classe de 52 milhões de dispositivos.

3.1. Discussão

Os trabalhos relacionados demonstram que *Web fingerprinting* é empregado tanto para identificar usuários quanto para identificar a classe de seus dispositivos. No que tange a identificação da classe, três métodos são utilizados: performance do JavaScript, conformidade do JavaScript e desenhos no HTML5 Canvas. Embora eficaz, o teste de performance não é eficiente, pois pode exigir recursos do navegador de forma a comprometer a experiência de navegação na *Web*, além do tempo de execução ser maior do que os de outros trabalhos encontrados na literatura. Já o teste de conformidade apresenta-se como eficaz e eficiente, mas é instável uma vez que um navegador (aprovado ou reprovado no teste) possa mudar de estado a medida que seja atualizado. O método que emprega HTML5 Canvas cumpre o objetivo, mas apenas recursos de vídeo foram explorados.

Ainda não há na literatura um método que emprega recursos de áudio para identificar classes de dispositivos.

4. Proposta e Implementação

Esta Seção apresenta a proposta do método e explica seus elementos, arquitetura e funcionamento.

4.1. Visão Geral

A Figura 2 ilustra, em alto nível, a visão geral da proposta.

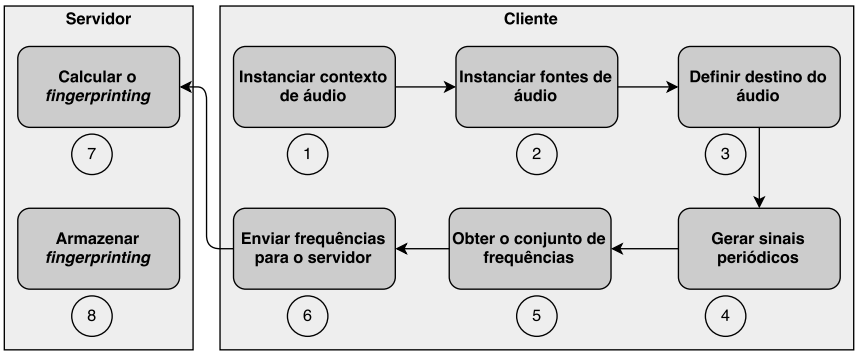


Figura 2. Visão geral de fluxo do método proposto.

O método proposto consiste em: Instanciar, no lado do cliente, um contexto de áudio **(1)**, criado a partir do *AudioContext* ou do *OfflineAudioContext*; Instanciar fontes de áudio **(2)**, que podem ser representadas por um oscilador (*OscillatorNode*) – capaz de gerar ondas periódicas (por exemplo, senoidal, quadrada e triangular) – ou por arquivos de áudio (música em mp3, por exemplo); **(3)** Definir o destino do áudio, que geralmente são os alto-falantes; **(4)** Gerar sinais periódicos – no caso do *OscillatorNode*, basta usar o método *start()* e no caso de arquivos de áudio é necessário fazer a decodificação dos mesmos; **(5)** Obter o conjunto de frequências (características) das ondas periódicas ou músicas, que pode ser obtido por métodos específicos ou pela leitura do *AudioBuffer* de saída; **(6)** Enviar as frequências para o servidor; **(7)** Calcular o *fingerprint* a partir das características extraídas, por exemplo, com o uso de funções de *hashing*; e, por fim, **(8)** Armazenar o *fingerprint* no banco de dados para análises de resultados.

Ao final do processo, as características obtidas como resultado serão enviadas para o servidor. Os passos 1 à 4 resultarão em sinais que serão processados. Parte do processamento consiste em calcular[1] a FFT (*Fast Fourier Transform*) dos mesmos, cujos resultados serão as características extraídas do dispositivo, capaz de identificar a classe do mesmo.

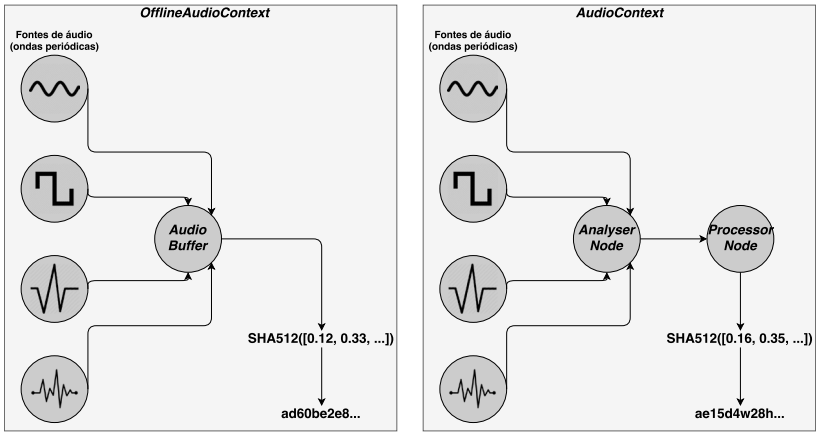
4.2. Implementação

A Figura 3 ilustra os fluxos de processamento de áudio que podem ser usados para implementar o método de *Web fingerprint* proposto.

No caso do *OfflineAudioContext* (Figura 3 à esquerda), quando uma onda é gerada, sua FFT é calculada e o resultado é passado para um *AudioBuffer*, onde o sinal periódico fica armazenado. Após todos os sinais estarem em seus respectivos *AudioBuffers*, as primeiras 2048 frequências são coletadas e enviadas para o servidor, onde será calculado o *fingerprint* com o emprego da função de *hashing* SHA512.

Quando se trata do *AudioContext*, o processo é similar, mas diferente em aspectos relacionados a topologia do fluxo de áudio. Quando os sinais periódicos são gerados, os mesmos são passados para o *AnalyserNode* e para o *ScriptProcessorNode*, onde a FFT

[1] A *Web Audio* API se encarrega de fazer isso.



(a) (b)

Figura 3. Arquitetura do método proposto a partir de dois pontos de vista: *OfflineAudioContext* (à esquerda) e *AudioContext* (à direita).

de cada sinal será calculada e os dados de análise dos sinais serão obtidos. Após este processo, um conjunto com 2048 elementos contendo os dados de análise é obtido e então enviado para o servidor, onde o *fingerprint* será calculado.

5. Experimentos

Essa Seção apresenta os experimentos que foram realizados para avaliar o método de *Web fingerprinting* proposto. Primeiramente, são descritos os experimentos preliminares realizados para avaliar qual interface (*AudioContext* ou *OfflineAudioContext*) é mais viável de se utilizar. Em seguida, é apresentado o experimento realizado com usuários voluntários.

5.1. *AudioContext* vs *OfflineAudioContext*

Uma vez que a *Web Audio API* possui duas interfaces disponíveis e, aparentemente, aptas a serem empregadas em um *Web fingerprinting*, qual delas utilizar? Com este questionamento em mente, decidiu-se realizar uma série de experimentos envolvendo os recursos disponíveis em cada uma das interfaces. Desta forma, foram definidos seis (6) cenários para experimentação, conforme a Tabela 2.

Tabela 2. 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[2] (Mac OS X 10.11). O processo de experimentação ocorreu da seguinte maneira: A página do experimento[3] 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), 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. Os navegadores utilizados foram: Chrome 59, Firefox 54, Edge 15, Safari 10 e Opera 46.

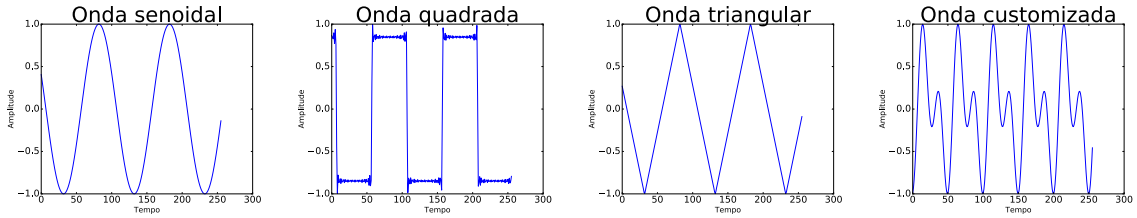


Figura 4. Ondas periódicas.

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 5 ilustra os resultado dos experimentos CE1 e CE2.

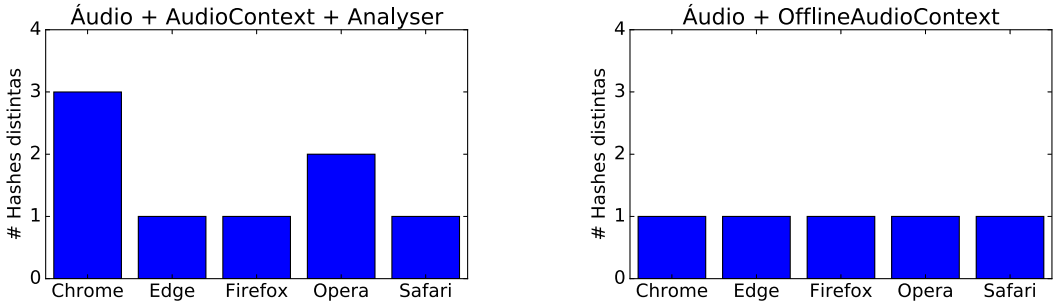


Figura 5. Experimentos com os cenários CE1 e CE2.

Percebe-se no primeiro cenário (Figura 5 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

[2]Devido a falta de disponibilidade de uma máquina física.

[3]Por ser um experimento controlado, a página ficou hospedada em um servidor local.

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 5 à direita). Isso serve de indicativo que o *OfflineAudioContext* pode contribuir com artefatos de *Web Fingerprinting*.

O experimento CE3 (Figura 6 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 6 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.

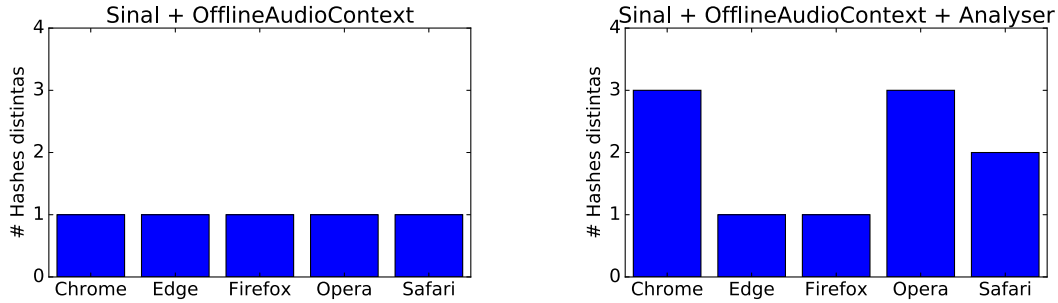


Figura 6. Experimentos com os cenários CE3 e CE4.

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 experimentos 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*.

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 7 à esquerda e direita, respectivamente. Por exemplo, na Figura 7 à esquerda, o Chrome apresentou 5 chaves distintas ao longo das 50 repetições. Na Figura 7 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*

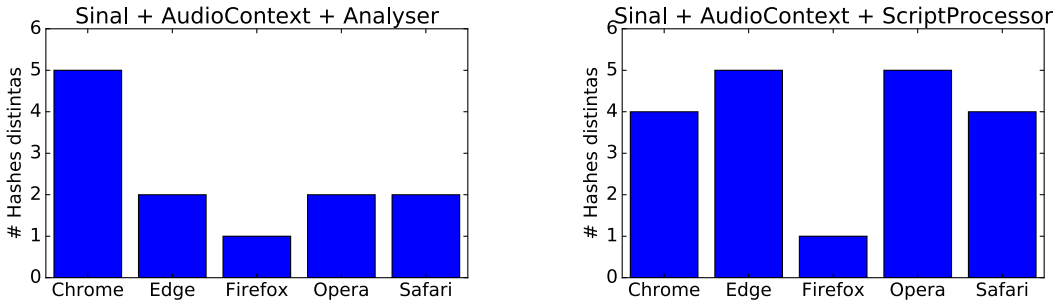


Figura 7. Experimentos com os cenários CE5 e CE6.

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 à processamentos podem ocasionar instabilidades [W3C 2017]. 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.

5.2. 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*[4] também é coletado.

Com relação as ondas periódicas, são empregadas as mesmas descritas na Subseção 5.1, ilustradas na Figura 4. A proposta está fundamentada no fato de que navegadores e dispositivos distintos geram a mesma onda, mas com pequenas diferenças no conjunto de frequências [Englehardt and Narayanan 2016]. 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 Seção 5.1. 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

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

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.

6. Resultados e Discussão

A Figura 8 ilustra algumas características dos dispositivos das pessoas que participaram do experimento, realizado por 11 dias (16 à 26 de junho de 2017).

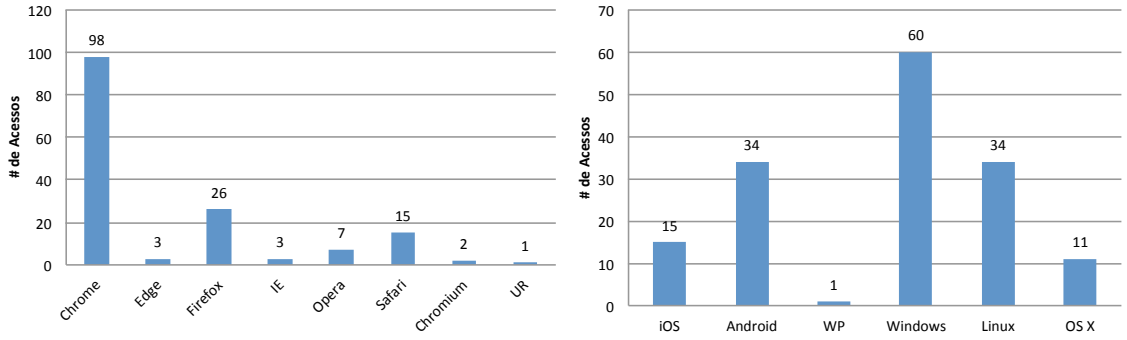


Figura 8. Características dos dispositivos usados no experimento.

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 a frente essa "definição" espontânea de qual tipo de dispositivo apresentará 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[5] 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 8 à direita). Também foi detectado um Windows Phone (WP).

Ao analisar as chaves identificadores geradas, percebeu-se um fato interessante. Houveram 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 3.

A primeira observação que se faz na Tabela 3 é que os navegadores que possuem o mesmo motor de renderização – caso do Chrome, Opera e Chromium – compartilham a mesma chave (linhas de 6 à 11). 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 da versão 51.0.2704.106 até 59.0.3071.109 na linha 6), sistema operacional e versão do sistema operacional. Por outro ponto de vista, o uso do *fingerprinting* baseado no *OffilineAudioContext* permite a identificação de navegadores distintos, ou seja, qual está sendo utilizado.

[5]https://www.ur-browser.com

Tabela 3. 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

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 são diferentes (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 o mesmo se encontra instalado. Esse resultado pode ser observado no navegador Chrome, versão 51.0.2704.106 nas linhas 6 e 22.

A terceira observação diz respeito a versão do navegador afetar no *fingerprint* gerado. Por exemplo, percebe-se que o Firefox 44 e 47 (linhas 44 e 4, respectivamente) 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.

Outro resultado interessante que foi observado durante a análise dos resultados é que os navegadores Safari, Chrome e Internet Explorer (linhas 45 à 51) estão agrupados, mesmo que seus respectivos motores de renderização sejam distintos. Acontece que para esses navegadores o conjunto de frequências retornado foi nulo devido a 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 defino e também a limitação do método proposto.

A quarta 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 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*.

7. Conclusão

Nesse trabalho foi apresentado um método de *Web Fingerprinting* que utiliza recursos providos pela *Web Audio API* (que até a data de escrita deste trabalho está em *working draft* [W3C 2017]) com a finalidade de identificar a classe dos dispositivos, por meio do emprego do *OfflineAudioContext*. Mostrou-se que com o *OfflineAudioContext* é possível gerar ondas periódicas, obter suas frequências e a partir disso gerar um *fingerprint*. Isso é possível devido a pequenas diferenças entre os navegadores e suas versões, bem como os tipos de dispositivos e os sistemas operacionais dos mesmos. Essas diferenças afetam em como o sinal é gerado, o que pode ser usado para identificar a classe dos dispositivos.

Os resultados apresentados na Seção 6 evidenciam a efetividade do método de *Web Fingerprinting* proposto nesta pesquisa, pois pode-se observar que o mesmo é capaz de identificar a classe dos dispositivos com base no seguinte conjunto de características: $\{$*Navegador; Versão do navegador; Tipo do dispositivo*$\}$. Embora não seja capaz de identificar unicamente os usuários, é possível perceber nuances do navegador e do dispositivo, demonstrando que a *Web Audio API* é uma das características relevantes a ser empregada em métodos de *Web Fingerprinting*.

7.1. Limitações e Trabalhos Futuros

Embora possa ser usado para identificar as classes dos dispositivos, o método de *Web Fingerprinting* proposto neste trabalho atribui chaves diferentes para dispositivos que possuem o mesmo conjunto $\{$*Navegador; Versão do Navegador; Tipo do dispositivo*$\}$, possivelmente devido a alguma característica que os distinguem, mas que esse estudo não foi capaz de detectar. Além disso, o método falha nos dispositivos e navegadores que não suportam a *Web Audio* API.

Como trabalhos futuros, pode-se destacar: (I) Investigar o por quê de dispositivos que na teoria deveriam fazer parte da mesma classe estão em classes distintas; (II) Investigar como o *AudioContext* ou *OfflineAudioContext* pode ser usado para identificar,

de forma única, os usuários, invés de apenas determinar a classe dos dispositivos; (III) Investigar o comportamento instável do *AudioContext* em métodos de *Web fingerprinting*.

Agradecimentos

Os autores deste trabalho agradecem à CAPES por viabilizar esta pesquisa.

Referências

Bursztein, E., Malyshev, A., Pietraszek, T., and Thomas, K. (2016). Picasso: Lightweight Device Class Fingerprinting for Web Clients. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 93 –102, Austria.

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

Englehardt, S. and Narayanan, A. (2016). Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1388–1401. ACM.

Khademi, A. F., Zulkernine, M., and Weldemariam, K. (2015). An Empirical Evaluation of Web-Based Fingerprinting. *IEEE Software*, 32(4):46–52.

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

MDN (2016). Web audio: conceitos e uso. `https://developer.mozilla.org/pt-BR/docs/Web/API/`.

Mowery, K., Bogenreif, D., Yilek, S., and Shacham, H. (2011). Fingerprinting Information in JavaScript Implementations. In *Proceedings of W2SP 2011*, pages 1–11. IEEE.

Mowery, K. and Shacham, H. (2012). Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of W2SP*, pages 1–12. IEEE.

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

Nakibly, G., Shelef, G., and Yudilevich, S. (2015). Hardware Fingerprinting Using HTML5. *CoRR*, abs/1503.0.

Saraiva, A., Feitosa, E., Elleres, P., and Carneiro, G. (2014). Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas. In *Livro de Minicursos do XIV SBSeg*, pages 49–98, Belo Horizonte - MG. SBC.

W3C (2017). Web Audio API. `https://webaudio.github.io/web-audio-api/`.

Ximenes, P., Correia, M., Mello, P., Carvalho, F., Franklin, M., and Andrade, R. (2016). TARP Fingerprinting: Um Mecanismo de Browser Fingerprinting Baseado em HTML5 Resistente a Contramedidas. In *Anais do XVI SBSeg*, pages 100–113, Niterói - RJ, Brazil.

