WBF Analyzer: Um Método para Detecção de Browser Fingerprinting em Páginas Web

Geandro Farias de Matos[1], **Eduardo L. Feitosa**[1]

[1]Instituto de Computação (IComp) - Universidade Federal do Amazonas (UFAM)
Manaus – AM – Brazil

geandro@ufam.edu.br, efeitosa@icomp.ufam.edu.br

Abstract. *The paper describes a method to detect fingerprinting calls to JavaScript objects in web pages and measure their severity level. The proposed method, called WBF Analyzer, was evaluated in different web pages databases. The results prove that previous work generated too many false positives and was unable to detect a set of JavaScript objects related to browser fingerprinting.*

Resumo. *Esta artigo descreve um método para detectar chamadas fingerprinting aos objetos JavaScript em páginas web e mensurar seu nível de severidade. O método proposto, chamado WBF Analyzer, foi avaliado em diferentes bases de páginas web. Os resultados provam que trabalhos anteriores geravam muitos falsos positivos e eram incapazes de detectar um conjunto de objetos JavaScript relacionados a browser fingerprinting.*

1. Introdução

As principais técnicas de *browser fingerprinting* utilizam Javascript para obter acesso a objetos internos do HTML DOM (*Document Object Model*) [Rausch et al. 2014] e, assim, conseguir informações relacionadas ao sistema operacional, ao hardware e ao próprio navegador. Embora a literatura enumere diferentes formas de detectá-las, os trabalhos que abordam a detecção de objetos Javascript que fornecem informações do navegador ou hardware (ligados assim a *fingerprinting*) são escassos.

Dentre os trabalhos propostos, a verificação de código Javascript realizada dinamicamente, através da avaliação, em tempo de execução, via *proxy* é a mais usual. O problema com esses trabalhos é que eles dependem da ocorrência de um ou mais eventos, geralmente em certas condições, para realizar a detecção. Além disso, requerem muito poder computacional para avaliar grandes quantidades de linhas de código. Já os trabalhos que realizam a verificação estaticamente, abordagens mais simples e rápidas, o fazem aplicando técnicas que checam o código Javascript de forma léxica, usando *regex*, por exemplo, para fazer a identificação de *fingerprinting*. Contudo, a avaliação do ponto vista léxico traz problemas, pois não se tem como definir, com precisão, se um conjunto de dados léxicos avaliados é realmente um objeto Javascript que pode ser chamado ou se é apenas uma *string* declarada no código.

Neste cenário, o objetivo deste artigo é identificar *browser fingerprinting*, através da observação e identificação das chamadas aos objetos Javascript fornecedores de informações, empregando análise estática através de Árvore Abstrata Sintática (*AST*), a fim de aumentar a detecção deste tipo de rastreamento e fornecer um arcabouço para futuras contramedidas. Especificamente, pretende-se: (i) elaborar um extrator para coletar o

código Javascript de páginas web; (ii) criar um dicionário de objetos da linguagem Javascript comumente utilizados em *fingerprinting*; (iii) desenvolver um identificador sintático, baseado em AST, para identificar chamadas aos objetos da linguagem Javascript. É importante destacar que a parte inicial desta pesquisa foi publicada como artigo curto no SBSeg 2020 [Matos and Feitosa 2020].

O restante deste artigo está organizado como segue: a seção 2 apresenta os conceitos necessários para a compreensão da proposta; a seção 3 descreve os trabalhos relacionados; a seção 4 apresenta o método proposto; a seção 5 apresenta a implementação; a seção 6 expõe os resultados; e, por fim, a seção 7, as conclusões e trabalhos futuros.

2. Conceitos Básicos

2.1. Browser Fingerprinting

A RFC (*Request For Comments*) 6973 [Cooper et al. 2013] define *fingerprint* como "um conjunto de elementos de informação que caracteriza um dispositivo ou uma instância de uma aplicação" e *fingerprinting* como "o processo pelo qual um observador ou atacante identifica, de maneira única e com alta probabilidade, um dispositivo ou uma instância de um aplicativo com base em um conjunto de múltiplas informações". [Saraiva et al. 2014] define *browser fingerprinting* como um conjunto de técnicas usadas para identificar (ou reidentificar) um usuário ou um dispositivo, através de um conjunto de configurações, atributos (tamanho da tela do dispositivo, versões de software instalado, entre muitos outros) e outras características observáveis durante as comunicações.

Tipicamente, *browser fingerprinting* é realizado via códigos escritos em linguagem Javascript, ou chamadas de objetos via Javascript, para obter as informações armazenadas nas variáveis de ambiente do navegador quando a página web é acessada. Dessa forma, reconhecer quais objetos de Javascript representam ações de *fingerprinting*, através da análise do código fonte da página web, e determinar o risco que a página oferece é fundamental.

2.2. Árvore Sintática Abstrata (AST)

Árvore Sintática Abstrata (AST) é uma estrutura de dados, baseada em árvore, que representa um código fonte a ser inspecionado de forma programática. Uma AST pode ser definida como uma representação intermediária do código fonte que favorece análises insensíveis ao fluxo, ignorando a ordem em que as instruções são executadas e possibilitando exploração de diversos caminhos a serem verificados. A Figura 1 ilustra uma AST com chamadas a objetos ligados a *fingerprint*.

Pode-se observar na figura que **console.log("navigator.userAgent")** é o nó raiz da AST. Ele possui uma chamada de expressão com duas arestas que apontam para outros nós. A aresta $.calle$ aponta para o objeto **console.log**, um nó do tipo $MemberExpression$, que aponta para o objeto **console** e a propriedade **log**, ambos do tipo $Identifier$. A aresta $.arguments$ aponta para o argumento dessa chamada. Nota-se, com essa análise simplificada, o emprego de AST para detecção de *browser fingerprinting*, uma vez que **navigator.userAgent** - termo ligado a *fingerprint* - nessa condição, não pode ser classificado como um, pois a chamada **console.log** apenas imprimirá a palavra **navigator.userAgent**.

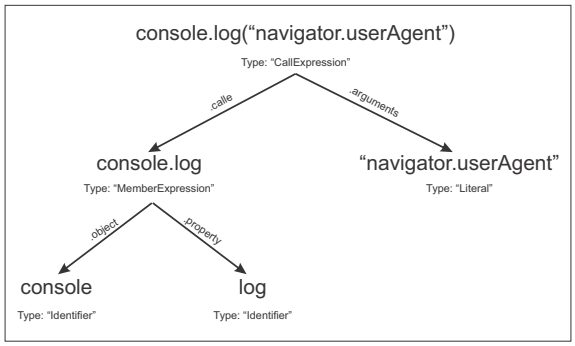

Figura 1. Exemplo de AST avaliando um elemento de *fingerprint*. Fonte: [Matos and Feitosa 2020]

É importante destacar que ao utilizar uma AST, pode-se obter um relatório completo de análise contendo a estrutura do código de forma hierárquica e rótulos que identificam essa estrutura, motivo pelo qual ASTs são usadas neste trabalho.

3. Trabalhos Relacionados

Esta seção discute somente trabalhos relacionados sobre *browser fingerprinting* que avaliaram o código Javascript de forma estática, foco desta pesquisa. No trabalho de [Rausch et al. 2014], os autores avaliaram o código Javascript para medir a frequência de certos scripts de *fingerprint* comumente usados para identificar os usuários. Os autores examinaram e compararam as amostras somente com três scripts de *browser fingerprinting* conhecidos. Contudo, esta análise demonstrou-se frágil, pois tentou identificar apenas scripts conhecidos e, segundo a pesquisa, menos que 6% dos sites examinados utilizam um desses scripts. A pesquisa de [Saraiva et al. 2016] avaliou o código Javascript para identificar *fingerprinting* usando *regex* e criou uma metodologia de classificação. Ela observou os aspectos léxicos do código Javascript, onde foi aplicada uma *regex* para contar palavras suspeitas encontradas no código estático. O problema com esta abordagem foi que o uso de *regex* gerou falsos positivos ao contar objetos como se fossem strings.

[Elleres et al. 2017] propôs um método capaz de detectar e avaliar scripts Canvas *fingerprinting* em páginas web, com base na extração de características relevantes do conteúdo estático do documento, por meio de técnicas de recuperação da informação. Em sua pesquisa, [van Vulpen 2020] empregou AST para criar uma ferramenta automatizada para separar código Javascript ligado a *fingerprinting* e, assim, identificar *fingerprinters* comerciais conhecidos. Como resultado, identificou 13 *fingerprinters* comerciais desconhecidos.

Diferente dos trabalhos exposto, esta pesquisa propõe identificar as chamadas aos objetos Javascript que atuam como indicadores de *fingerprinting* em páginas web, através da identificação estática baseada em AST. Tomando como base os trabalhos citados, a AST é uma técnica de identificação que permite extrair informações do código com precisão. Além do mais, é uma técnica que já foi avaliada em outras pesquisas e por este motivo fornece uma base sólida de análises. Assim, pode-se avaliar o fluxo do código rotulado com as características estruturais da linguagem onde se aplica um *parser* baseado em AST. Também cabe destacar que o foco desta pesquisa é idenficar toda e qualquer chamada a objetos Javascript ligada a *fingerprinting* e não algumas especificidades.

4. WBF Analyzer

A Figura 2 ilustra a visão arquitetural da **WBF (*Web Browser Fingerprinting*) Analyzer**. Seus componentes são detalhos a seguir.

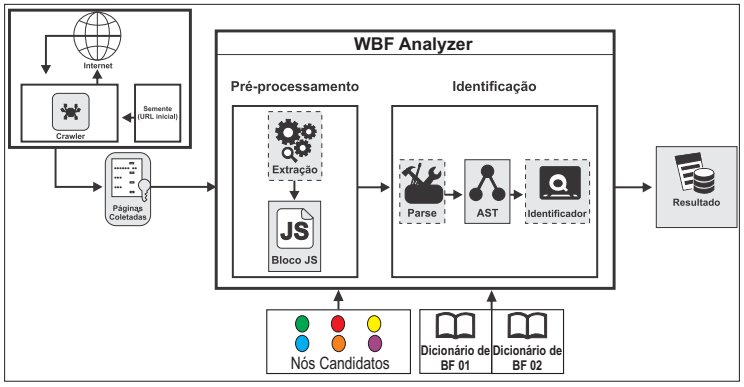

Figura 2. Arquitetura da WBF Analyzer. Fonte: [Matos and Feitosa 2020]

Para funcionar, a **WBF Analyzer** precisa dos códigos Javascript das URLs que serão analisadas. Esse processo pode ser feito por um *web crawler*. Assume-se que todas as URLs são verificadas, quanto a sua validade (por exemplo, se estão mal-formadas, duplicadas ou com ausência de objetos Javascript).

4.1. Pré-Processamento

A primeira etapa do **WBF Analyzer** é o pré-processamento do código Javascript, que basicamente contempla uma fase de extração que verifica e limpa o código do processo de ofuscação.

Em linhas gerais, seu funcionamento é o seguinte: após receber como entrada a página coletada, é feita uma verificação dentro das tags $< html >< /html >$ para encontrar as subtags $< script >< /script >$, $< script\ src >< /script >$. A primeira tag $< script >< /script >$ fornece os códigos Javascript que estão "in-line" na página, ou seja, foram declarados diretamente no corpo do documento *HTML*. Já a segunda tag $< scriptsrc >< /script >$ fornece, através do atributo src, a localização dos scripts que foram declarados externos ao corpo do documento *HTML*. Em seguida, ocorre a extração correta do código Javascript. Além disso, tem-se nessa fase a limpeza da ofuscação, aplicada para que o código torne-se legível.

Ao final do tratamento, o código Javascript é separado do HTML e limpo da ofuscação, produzindo como saída um bloco Javascript formado apenas por códigos "in-line" e externos que será enviado para a etapa de Identificação.

4.2. Identificação

Na etapa de identificação, o bloco Javascript é recebido e um *parser* o transforma em uma AST, que é então repassada para o identificador. Após o identificador receber a AST como entrada, ele aplicará três (3) regras necessárias para detecção das chamadas de *browser fingerprinting*.

A Figura 3 ilustra o identificador recebendo a AST, bem como os nós candidatos de onde serão extraídos os termos que irão gerar um escopo reduzido da AST. Também

recebe como entrada um dicionário de *browser fingerprinting*, que está dividido em dois dicionários (*Browser Fingerprinting* 01 e 02), onde o 01 é utilizado como ferramenta de suporte na normalização das chamadas e o 02 como ferramenta de suporte na classificação das chamadas.

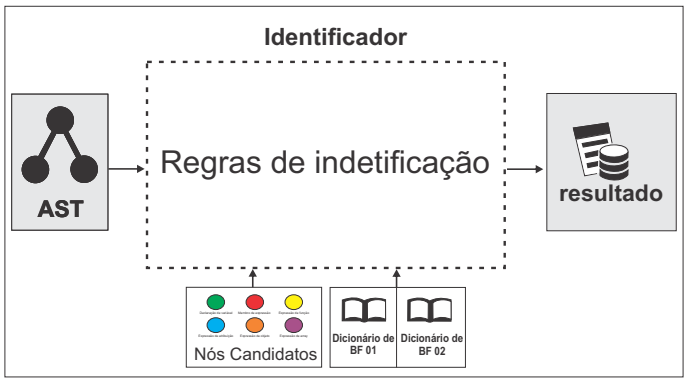

Figura 3. Identificador.

O funcionamento do identificador é baseado na execução das três (3) regras de identificação. A **Regra 1 - Extração** percorrer a(s) AST(s) de entrada para identificar e realizar a extração de termos. A varredura é feita de forma insensível ao fluxo para garantir que todos os caminhos possíveis da AST sejam examinados. Os resultados dessa verificação e extração são armazenados em um escopo reduzido, gerado para evitar que se façam manipulações que envolvam a edição da AST. O escopo reduzido (Figura 4) é uma estrutura de dados formada pelo identificador esquerdo (operando esquerdo de uma expressão de atribuição), identificador direito (operando direito de uma expressão de atribuição); objeto 0, objeto 1 e objeto 2 (objetos Javascript); e propriedade 1 e propriedade 2 (baseados na estrutura do dicionário de *fingerprinting* proposto em [Saraiva et al. 2016]).

Identificador Esquerdo	Identificador Direito	Objeto 0	Objeto 1	Objeto 2	Propriedade 1	Propriedade 2
b		navigator			userName	

Figura 4. Escopo reduzido

Por exemplo, o código $b = navigator.userName$ teria a váriavel b atribuída ao identificador esquerdo, $navigator$ para o objeto 0 e $userName$ para a propriedade 1. Os outros elementos do escopo receberiam 0. Com isso, pode-se armazenar os dados extraídos da AST em uma estrutura de dados maleável como, por exemplo, uma lista encadeada. Assim, evita-se a perda de dados na AST e evita-se processos custosos como a inserção de registradores e de marcadores nos nós da AST.

A **Regra 2 - Normalização** é responsável por normalizar a relação entre objetos e propriedades para produzir as chamadas que dão origem ao escopo normalizado. Essa regra recebe como entrada um escopo reduzido, onde são aplicados uma série de processos (verificação, comparação e correspondência). A normalização possui como saída um escopo normalizado (Figura 5), uma estrutura de dados que armazena os dados para contagem de frequência e classificação das chamadas, onde o campo linha armazena a origem da chamada encontrada no escopo reduzido, o campo qtd armazena a quantidade de vezes que a chamada foi encontrada e o campo chamada armazena o nome da chamada.

Linha	Qtd	Chamada
1	1	navigator.plugins.name

Figura 5. Escopo normalizado

A **Regra 3 - Classificação** é responsável por aplicar a contagem de frequência, a classificação de risco da chamada e a classificação de risco da página web. A regra 3 recebe como entrada o escopo normalizado e produz como saída o escopo classificado, uma estrutura de dados de saída que registra a frequência que uma chamada aparece no código fonte, o nome do objeto chamado, a classificação de risco da chamada e a classificação da página. Com a aplicação da regra 3, pode-se definir qual o risco de *Browser fingerprinting* que uma página web representa para o usuário. Após essa avaliação, a classificação das chamadas é realizada em nível baixo, médio e alto, segundo o método de [Saraiva et al. 2016].

Uma explicação detalhada e completa sobre o funcionamento das três regras, suas estruturas e processos pode ser obtida no artigo "Identificando Indicadores de *Browser Fingerprinting* em Páginas Web" [Matos and Feitosa 2020].

5. Implementação

No âmbito geral, a **WBF analyzer** foi implementada na plataforma *Node.js* para permitir a execução de códigos Javascript fora de um navegador web. A biblioteca *Esprima* foi empregada para realizar as análises léxicas e sintáticas de códigos Javascript, transformando-os em uma representação intermediária e gerando uma árvore abstrata sintática do código fonte. A linguagem de desenvolvimento python foi usada para aproveitar os recursos voltados para ciência de dados.

Foi elaborada uma implementação de um (*web crawling*) com objetivo de extrair códigos Javascript "in-line" e externos. Para tanto, as bibliotecas *urllib3* e *BeautifulSoup* foram utilizada. Para o pré-processamento foram utilizadas as bibliotecas *BeatifulSoup*, para retornar apenas strings; *re*, que usa uma regex para remover elementos específicos; e *requests*, que permite obter o conteúdo dos links para códigos Javascript "in-line".

Para mais detalhes sobre a implementação, pode-se consultar "endereço ocultado para revisão", onde todo o código está comentado. Pode-se também consultar os arquivos leiame.txt para mais detalhes sobre cada parte da aplicação.

6. Experimentos e Resultados

Todos os experimentos foram executados em uma máquina Intel(R) Xeon(R) CPU E5-4617 @ 2.90GHZ (2 processadores), com 90 GB de memória RAM, 100 GB de disco, com sistema operacional Windows 10.

6.1. Base de Dados

Para realização dos experimentos, foram utilizadas quatro (04) bases de dados. A primeira base, chamada de Alexa (Ano 2016), contém 2.002 sites (códigos) utilizados no trabalho de [Saraiva et al. 2016]. Esta base foi selecionada para verificar se a **WBF analyzer** pode identificar padrões ou quantidades de indicadores de *fingerprinting* não

identificados ou quantificados por [Saraiva et al. 2016]. A segunda base, Top 50 Alexa (Ano 2021)[1], contém códigos de 50 sites coletados especificamente das páginas principais de cada um dos sites do Top 50 da Alexa.com, na categoria internacional. Os códigos foram coletados via *web crawling* pela **WBF Analyzer**. Esta base foi montada para verificar se existe uma linha de crescimento de indicadores de *fingerprinting* apresentados em [Saraiva et al. 2016]. A terceira base, Canvas (Ano 2017), é oriunda do trabalho de [Elleres et al. 2017] e é composta por 8.350 códigos de sites. Esta base foi selecionada especificamente para verificar se os indicadores de *fingerprinting* que predominam são de *Canvas Fingerprinting* como mencionado em [Elleres et al. 2017]. A quarta e última base, $DMOZ$ (http://DMOZ.org/), também oriunda do trabalho de [Saraiva et al. 2016], contém 1.564 sites. Esta base foi selecionada por ser considerada no trabalho de [Saraiva et al. 2016] como benigna.

Vale destacar que as comparações dos resultados nas bases investigadas, sempre que for possível, serão feitas com o trabalho de [Saraiva et al. 2016], pelas seguintes razões: (i) é focado em ocorrências e chamadas de objetos Javascript e não limitou-se a um único objeto; (ii) avalia o código Javascript in-line e o código Javascript externo; (iii) criou uma metodologia de classificação de risco, a qual foi adaptada para ser utilizada na **WBF Analyzer**. Este trabalho não questiona a metodologia de risco estabelecida e sim o método utilizado para detectar as chamadas de Javascript.

6.2. Análise das Bases

Antes de apresentar a análise das bases de dados, é preciso deixar claro o que será descrito. Para todas as base será exposto o resultado da análise da presença de *fingerprinting* feita pela **WBF Analyzer**. Em seguida é apresentada uma análise da classificação de risco (de acordo com a metodologia proposta por [Saraiva et al. 2016]. Depois são apresentados os valores percebidos pela **WBF Analyzer** sobre chamadas e invocações de métodos *fingerprinting*. Por fim, é apresentado o resultado do escopo classificado, exibindo as chamadas mais relevantes.

6.2.1. Base Alexa (Ano 2016) - Resultado

A base Alexa.com (Ano 2016) possui um total de 2.002 sites. Deste total, 83.8% (1.678) foram considerados como contendo *fingerprinting*, isto é, a aplicação conseguiu detectar as chamadas de objetos Javascript neste sites. Já 15.5% (310) não foram analisados pela aplicação devido ao fato da árvore abstrata do código não ser gerada por erros na sintaxe do código ou *tokens* não reconhecidos pela biblioteca Esprima. Por fim, em 0.7% (14) não foram encontradas chamadas de objeto Javascript. Ressalta-se que o trabalho de [Saraiva et al. 2016] diz ter analisado os 2.002 sites. Também destaca-se que a redução na quantidade de sites avaliados não gera qualquer imprecisão no processo de avaliação, mas pode tornar a comparação com o trabalho de [Saraiva et al. 2016] mais enviesada.

Quanto a classificação de risco, os resultados da **WBF Analyzer** foram comparados com o trabalho de [Saraiva et al. 2016]. A Tabela 1 ilustra essa análise comparativa.

A **WBF Analyzer** conseguiu classificar melhor os sites, sendo 78.7% (1.320 sites) com nível de risco alto, 16.7% (280 sites) com nível de risco médio e 4.6% (78 sites) com

[1]http://www.alexa.com/

Tabela 1. Análise comparativa da classificação da base Alexa (Ano 2016)

Autores	Riscos		
	Baixo	Médio	Alto
[Saraiva et al. 2016]	1%	40%	59%
WBF Analyzer	4.6%	16.7%	78.7%

nível de risco baixo. Os resultados seguem a mesma tendência apresentada no trabalho de [Saraiva et al. 2016], embora, obviamente, os percentuais de classificação se apresentam maiores. Vale destacar que o **WBF Analyzer** consegue a classificação através do escopo classificado, invidualizado para cada site. Já em [Saraiva et al. 2016], o trabalho juntava todos os objetos e classificava os sites em relação aos níveis que cada objeto pertencia.

Sobre as chamadas e invocações, a **WBF Analyzer** as identifica através do formato $objeto.propriedade$, quantifica sua frequência (número de vezes que aparece no código) e as organiza no escopo classificado da base ou do site analisado. Os critérios estabelecidios para diferenciar chamadas de objetos de invocações de objeto são: (1) se um objeto chama sua propriedade através da notação ponto (.), ela contabiliza como uma chamada encontrada; (2) se um objeto chama uma propriedade que já foi chamada, ela contabiliza essa repetição no código como a frequência de aparição da chamada.

Desta forma, a **WBF Analyzer** encontrou 157 chamadas (objetos) na base Alexa.com (ano 2016). Deste total, 95 chamadas foram classificadas como nível baixo, 43 como nível médio e 19 como nível alto. Ao todo observou-se 157.530 invocações de chamadas de objetos Javascript dos 1.678 sites analisados. Desse total, 131.859 são chamadas para objetos Javascript classificados com risco baixo. 19.295 são chamadas para o objetos de risco médio e 6.376 são objetos de risco alto.

Embora o trabalho de [Saraiva et al. 2016] afirme ter encontrado 169.902 invocações na base, vale destacar que a **WBF Analyzer** faz sua análise sem aplicar a formação do bloco JS (função que unifica o código da página principal com o código de páginas que são dependentes do domínio principal, ou seja, junta código "in-line" e com os externos da página principal). Isto foi feito por não se saber como foram feitas as análises de [Saraiva et al. 2016]. Assim, o **WBF Analyzer** analisou todos os arquivos da base como arquivos independente, gerando escopos classificados para cada site da base. Os escopos classificados da base foram unidos posteriormente para quantificar as chamadas encontradas e suas invocações, e poder estabelecer uma classificação de risco para a base.

Por fim, como forma de provar a importância e a eficácia do Escopo Classificado, a Tabela 2 lista as maiores chamadas (em frequência) encontradas. Vale lembrar que: (i) chamada é onde ficam armazenadas todas as chamadas de objeto no formato $objeto.propriedade$ encontradas; (ii) frequência é onde fica registrado o número de vezes que uma chamada é invocada; e (iii) risco da chamada é onde fica registrado o risco da chamada, podendo ser baixo, médio ou alto.

6.2.2. Base Top 50 Alexa (Ano 2021) - Resultados

A base Top 50 Alexa (Ano 2021) é composta pelos 50 sites, listados na Alexa.com, no ano de 2021 e na categoria de sites internacionais. Nesta base, foram extraídos apenas os códigos Javascript das páginas principais de cada site. Como resultado, apenas um site

Tabela 2. Maiores chamadas no Escopo Classificado

Chamada	Frequência	Risco da chamada	Chamada	Frequência	Risco da chamada
this.width	36651	Baixo	this.height	32204	Baixo
new Date().getTime()	20289	Baixo	navigator.userAgent	14170	Baixo
document.cookie	3556	Médio	window.innerWidth	3115	Baixo
this.video	2843	Médio	document.referrer	2487	Médio
window.innerHeight	2423	Baixo	canvas.getContext()	2355	Alto

(que estava em manutenção) não teve indicadores de *fingerprinting*.

Quando avaliando a classificação de risco, a **WBF Analyzer** conseguiu categorizar 81.6% (40) dos sites como de risco alto, 8.2% (4) dos sites como de risco médio e 10.2% (5 sites) como de risco baixo. Diferente da base Alexa.com (Ano 2016), o resultado desta base não pode ser comparado com o trabalho de [Saraiva et al. 2016]. Apenas como registro, a Tabela 3 lista os 50 sites avaliados e seu risco atribuído.

Tabela 3. Sites e risco da base Top 50 Alexa

#	Site	Risco	#	Site	Risco	#	Site	Risco
1	pandas.Tv	Nada encontrado	2	bongacams.com	Alto	3	intl.alipay.com	Médio
4	login.microsoftonline.com	Baixo	5	facebook.com	Alto	6	outlook.live.com	Alto
7	stackoverflow.com	Alto	8	weibo.com	Alto	9	vk.com	Alto
10	www.17ok.com	Baixo	11	twitter.com	Alto	12	www.360.cn	Alto
13	www.aliexpress.com	Alto	14	www.amazon.co.jp	Alto	15	www.adobe.com	Alto
16	www.amazon.com	Alto	17	www.amazon.in	Alto	18	www.baidu.com	Alto
19	www.aparat.com	Alto	20	www.bing.com	Baixo	21	www.csdn.net	Alto
22	www.google.com.hk	Alto	23	www.ebay.com	Alto	24	www.huanqiu.com	Alto
25	www.instagram.com	Baixo	26	www.google.com	Alto	27	www.jd.com	Alto
28	www.linkedin.com	Alto	29	www.naver.com	Alto	30	www.microsoft.com	Alto
31	www.netflix.com	Médio	32	www.office.com	Alto	33	www.okezone.com	Alto
34	www.qq.com	Alto	35	www.reddit.com	Baixo	36	www.shopify.com	Alto
37	www.sina.com.cn	Alto	38	www.sohu.com	Alto	39	www.tianya.cn	Alto
40	www.taobao.com	Alto	41	www.twitch.tv	Alto	42	www.wikipedia.org	Médio
43	www.tmall.com	Alto	44	www.yahoo.co.jp	Alto	45	www.yahoo.com	Alto
46	www.yy.com	Alto	47	www.zhanqi.tv	Médio	48	xinhuanet.com	Alto
49	zoom.us	Alto	50	www.youtube.com	Alto			

Sobre chamadas e invocações, foram detectadas um total de 125 chamadas de objetos e 5.592 invocações. Destas, 80 chamadas e 3.800 invocações foram classificadas como de risco baixo, 34 chamadas e 1.491 invocações como de risco médio e 11 chamadas e 301 invocações como risco Alto.

Por fim, sobre escopo classificado, notou-se que o objeto $date()$, invocando a propriedade $getTime()$, tem o maior percentual, o que nos fornece um indicador da tendência de *fingerprinting* para data e hora do computador. Percebe-se também a soma dos valores das chamadas $this.width$ e $this.height$ totaliza 911 invocações, superando as 875 da chamada de objeto $Date()$. Este também é um indicativo de que nesta base existe uma tendência de *fingerprinting* para largura e altura da tela do usuário. Por fim, outro ponto percebido é que existe predominância para os objetos classificados com risco baixo.

6.2.3. Base Canvas (Ano 2017) - Resultados

A base Canvas (Ano 2017) possui um total de 8.350 sites. Deste, 64.1% (5.350) dos sites foram considerados como contendo *fingerprinting*. Já 11.1% (927) dos sites não foram analisados. Por fim, em 24.8% (2.073) dos sites não foram encontradas chamadas de

objeto Javascript. Sobre a classificação de risco, a **WBF Analyzer** conseguiu categorizar 5.330 sites com os seguintes riscos: 84.8% (4.519 sites) deles com nível de risco alto, 13.0% (696 sites) com nível de risco médio e 2.2% (118 sites) com nível de risco baixo.

Foram detectadas um total de 175 chamadas de objetos e 729.801 invocações. Destas, 18 chamadas e 557.531 invocações foram classificadas como de risco baixo, 47 chamadas e 124.175 invocações como de risco médio e 18 chamadas e 48.095 invocações como risco Alto. Fazendo uma comparação apenas de valores, o trabalho de [Saraiva et al. 2016] encontrou 45.626 invocações nessa mesma base. Já o **WBF Analyzer** encontrou 729.801 invocações. Essa diferença ocorre porque [Saraiva et al. 2016] não resolve o problema da ofuscação de *string de substituição de palavra chave*.

Por fim, sobre o escopo classificado, a Tabela 4 lista as maiores chamadas (em frequência). Foram encontradas 124.832 invocações para o objeto $newDate().getTime()$, que pertence ao nível baixo. É interessante notar que os objetos do nível baixo e médio são mais costumeiramente utilizados para coletar informações e produzir chaves identificadoras do usuário do que os objetos vinculados ao nível alto.

Tabela 4. Maiores chamadas encontradas no escopo classificado

Chamada	Frequência	Risco da chamada	Chamada	Frequência	Risco da chamada
new Date().getTime()	124832	Baixo	this.width	99257	Baixo
this.height	82544	Baixo	navigator.userAgent	79271	Baixo
document.referrer	27487	Medio	document.cookie	22468	Medio
window.innerWidth	17387	Baixo	window.innerHeight	16137	Baixo
canvas.getImageData()	13673	Alto	window.localStorage	13398	Médio
screen.width	11189	Baixo	**canvas.getContext()**	10596	Alto
this.video	9720	Medio	screen.height	9583	Baixo
window.screen.width	9117	Baixo	window.screen.height	8616	Baixo
this.history	8421	Alto	**canvas.toDataURL()**	8215	Alto

A **WBF Analyzer** conseguiu identificar as chamadas de objetos principais desta base, segundo o trabalho de [Elleres et al. 2017] que também a analisou. Elas estão destacas em negrito na Tabela, representando 13.673 invocações para o objeto $canvas.getImageData()$, 10.596 invocações para o objeto $canvas.getContext()$, 8.215 invocações para o objeto $canvas.toDataURL()$. O objeto $canvas.toDataURL()$ é o objeto potencialmente mais perigo citado por [Saraiva et al. 2016], mas o menos solicitado.

6.2.4. Base DMOZ (Ano 2016) - Resultados

A base DMOZ (Ano 2016) possui um total de 1.564 sites. Deste, 83.0% (1298) dos sites foram considerados como contendo *fingerprinting*. Já 5.1% (79) dos sites não foram analisados. Por fim, em 12.0% (187) dos sites não foram encontradas chamadas de objetos.

Na classificação de risco, a **WBF Analyzer** conseguiu classificar os sites com a seguinte proporção: 59.2% (769) deles com nível de risco alto, 26.0% (338) com nível de risco médio e 14.7% (191) com nível de risco baixo. Os resultados obtidos são divergentes daqueles do trabalho de [Saraiva et al. 2016]. A **WBF Analyzer** classificou mais sites com risco alto, enquanto o trabalho de [Saraiva et al. 2016] classificou mais com riscos baixo e médio. A Tabela 5 ilustra essa análise comparativa.

Já sobre chamadas, foram encontradas 93 chamadas para o nível baixo, 42 chamadas para o nível médio e 15 chamadas para o nível alto, totalizando 150 chamadas

Tabela 5. Análise comparativa da classificação da base DMOZ (Ano 2016)

Autores	Riscos		
	Baixo	Médio	Alto
[Saraiva et al. 2016]	20.0%	47.0%	33.0%
WBF Analyzer	14.7%	26.0%	59.2%

encontradas. No quesito invocações, observaram-se 81.920 invocações de chamadas de objetos Javascript dos 1.298 sites analisados. Desse total, 65.775 são chamadas para objetos Javascript classificados com risco baixo. 13.101 são chamadas para o objetos de risco médio e 3.044 são objetos de risco alto.

Como comparação, enquanto [Saraiva et al. 2016] identificou 16.621 invocações de chamadas ligadas a *fingerprinting* ao avaliar 1.584 sites, a **WBF Analyzer** conseguiu identificar 81.920 invocações de chamadas ligadas a *fingerprinting* ao avaliar 1.298 sites. Os resultados demonstram que o método proposto neste trabalho é mais eficiente para identificar chamadas de objetos do que o método proposto por [Saraiva et al. 2016]. É preciso contextualizar que os valores apresentados na Tabela são apenas referente a comparação de contagem de chamadas, uma vez que [Saraiva et al. 2016] também contabilizou aparições isoladas dos objetos ou nomes de objetos usados com referência para a utilização de variáveis.

A Tabela 6 ilustra uma comparação de classificação de risco dos sites avaliados entre a **WBF Analyzer** e o trabalho de [Saraiva et al. 2016].

Tabela 6. Ocorrência de chamadas encontradas na base DMOZ

websites	Nível 1 - Baixo					Nível 2 - Médio					Nível 3 - Alto			
	innerWidth	innerheight	userAgent	screen.height	screen.width	referrer	cookie	mimeTypes	plugins	localStorage	getUserMedia	getImageData()	toDataURL()	History
robertkleingallery.com	0	0	0	2	2	0	0	0	0	0	0	0	0	0
pranapoweryoga.com	1	0	3	**3**	**3**	0	0	0	0	0	0	0	0	0
joegreenephoto.com	1	5	3	**2**	**1**	0	0	0	0	0	0	0	0	0
captel.com	0	0	0	1	1	0	0	0	0	0	0	0	0	0
dynamicpost.co.uk	**3**	5	1	**1**	**1**	0	0	0	0	0	0	0	0	0
oldogpaws.com	0	0	**4**	**3**	**4**	1	**11**	0	**5**	0	0	0	0	**3**
ilight-tech.com	**8**	**3**	**14**	**28**	**29**	1	**1**	**1**	**5**	0	0	0	0	0
bbc.com	**9**	**6**	11	**11**	**13**	**13**	**62**	**9**	**12**	**1**	0	0	0	0
gourmetsleuth.com	4	4	18	6	**10**	12	6	0	0	**8**	0	0	0	**4**
nordicademy.com.au	4	4	**3**	**24**	**24**	0	**2**	3	5	0	0	0	0	0
nikonusa.com	**11**	**9**	**41**	**399**	**400**	5	42	6	31	**16**	0	0	0	**23**
flairstrips.com	**4**	**2**	**25**	**55**	**73**	0	**2**	**1**	**10**	**2**	0	6	1	**0**
robotshop.com	**4**	**6**	**24**	**34**	**31**	**5**	**10**	**0**	**7**	**8**	0	3	1	1
fibergarden.com	**5**	**8**	**15**	**25**	**25**	**3**	**9**	**0**	**1**	**9**	0	3	1	**0**
newscooter4less.com	**4**	**2**	**1**	**13**	**14**	0	0	**0**	0	0	0	**0**	1	**0**

Os resultados na cor verde são aqueles onde a **WBF Analyzer** foi mais eficiente. Na cor preta os resultados em que houve empate e na cor vermelha os resultados em que a metodologia de detecção de [Saraiva et al. 2016] foi superior. Assim, nos resultados da comparação direta no total houve 116 empates, 46 vitórias e 44 derrotas para a **WBF Analyzer**.

No escopo classificado, a Tabela 7 enumera as 10 chamadas mais encontradas.

Tabela 7. Chamadas mais encontradas no Escopo Classificado

Chamada	Frequência	Risco da chamada	Chamada	Frequência	Risco da chamada
this.width	15238	Baixo	this.height	13081	Baixo
new Date().getTime()	12281	Baixo	navigator.userAgent	8125	Baixo
document.cookie	3126	Médio	document.referrer	2048	Médio
window.innerheight	1516	Baixo	this.video	1445	Médio
window.innerWidth	1403	Baixo	screen.width	1132	Baixo

Observou-se na base DMOZ que os sites tendem a fazer *fingerprinting* baseados em um conjunto de objetos que retornam informações a respeito largura da tela, da altura da tela, data e hora, das informações do navegador e correlação de navegação do usuário. Percebe-se também que dos cinco objetos mais invocados, quatro estão ligados ao nível baixo e apenas um objeto está ligado ao nível médio. Não foram retornados entre os cinco objetos mais populares da base objetos ligados ao nível alto.

6.3. Discussão

Comparando o resultados obtidos nas 4 bases de dados, quanto a classificação de risco, percebe-se que a Canvas (Ano 2017) e a Top 50 Alexa (Ano 2021), com 84,8% e 81,6%, respectivamente, apresentaram os maiores riscos. Contudo, os resultados da Canvas demonstraram que, apesar de ser considerada por [Elleres et al. 2017] como potencialmente voltada para o objeto *Canvas*, ela tem predominância de chamadas com referência ao nível baixo. Além disso, verificou-se que o indicador predominante de *fingerprinting* é a chamada de objeto Javascript $newDate().getTime()$, com 124.832 invocações contra 10.596 invocações da chamada $canvas.getContext()$, 8.215 invocações da chamada $canvas.getImageData()$ e 13.673 invocações da chamada $canvas.toDataURL()$.

A base Top 50 Alexa nos mostra que existe uma linha de crescimento na coleta de objetos Javascript, acentuada por sites que, em tese, não oferecem risco algum para os usuários. Na base, identificou-se que 81.6% dos sites apresentam risco alto. Os resultados obtidos para a base Alexa (Ano 2016) comprovaram que a **WBF Analyzer** identificou um padrão de chamada que utiliza a palavra reservada $this$ em Javascript para acessar propriedades de objetos como em $this.userAgent$, $this.width$, $this.height$ e outros. Esse padrão não foi documentado nos trabalhos de [Saraiva et al. 2016, Elleres et al. 2017, van Vulpen 2020]. Por fim, na base DMOZ observou-se que a **WBF Analyzer** conseguiu identificar 12,2% de risco alto a mais que a metodologia de [Saraiva et al. 2016].

Discutindo mais sobre a classificação de indicadores de *browser fingerprinting*, a Tabela 8 lista o conjunto de objetos dominantes nas 4 bases analisadas. O objeto $this$, chamando a propriedade $width$, ocorreu 36.651 vezes para a análise da base Alexa (Ano 2016) enquanto que em [Saraiva et al. 2016] não foi mencionada nem uma ocorrência dessa chamada de objeto significando que este objeto não foi detectado. O mesmo fato ocorre para o objeto $this$, chamando a propriedade $height$. Na base Top 50 Alexa, observa-se que, apesar de terem sido avaliados apenas 49 sites, foi identificado o conjunto de objetos que mais se destacaram, o mesmo observado na análise da base Alexa (Ano 2016). Além disso, existe a predominância de *fingerprinting* de data e hora.

Quanto aos resultados da base Canvas, apesar do objeto $Canvas$ ter sido detectado pela **WBF Analyzer**, ele não se destacou o suficiente para compor o conjunto dos 5

Tabela 8. Objetod dominantes de *browser fingerprinting*

Bases	Nível Baixo				Nível Médio	
	new Date().getTime()	this.width	this.height	navigator.userAgent	document.cookie	document.referrer
Alexa (Ano 2016)	20289	36651	32204	14170	3556	—-
Top 50 Alexa (Ano 2021)	875	478	423	419	430	—-
Canvas (Ano 2017)	124832	99257	82544	79271	—-	27478
DMOZ (Ano 2016)	12281	15238	13081	8125	3126	

objetos mais coletados da base. O objeto $document.referrer$ foi o que mais destacou, com maior número de invocações. Essa base foi a única a apresentar um conjunto de objetos diferente das demais. Quanto ao indicador de *fingerprinting*, os de data e hora foram coletados 124.832 vezes. A base DMOZ foi a que apresentou menor coleta de informações em comparação com as outras bases, tendo como indicador principal o objeto $this$, chamando a propriedade $width$, para *fingerprinting* de tela.

Para finalizar, é possível afirmar que o objetivo desse artigo foi alcançado, uma vez que os resultados em base reais comprovam que o método desenvolvido para detectar, quantificar e classificar chamadas de objetos Javascript pode produzir resultados que favoreçam a análise da tendência de crescimento de coleta de dados. Vale destacar que todas as bases usadas nos experimentos, em sua forma desofuscada, estão disponíveis[2].

7. Conclusão

Este trabalho propôs um método para detectar chamadas em códigos de páginas web que podem fornecer um indicativo do estado atual da coleta de dados via *fingerprinting* de objetos de Javascript. Como forma de provar a eficiência do método, várias bases de páginas web, referente aos ano de 2016, 2017 e 2021, foram testadas e os resultados mostram que existe um conjunto de objetos de Javascript formado por *new Date().getTime(), this.width, this.heigt navigator.userAgent, document.cookie* relacionados a *browser fingerprinting*, que persistiu nas 5 primeiras colocações em todas as bases avaliadas. Esses indicadores demonstram que bloquear a coleta de informações que possam gerar um identificador único do usuário nos tempos atuais é uma tarefa difícil, pois esses objetos são comumente utilizados em páginas web para fins benéficos como ajustar as preferências de navegação do usuário.

7.1. Contribuições Alcançadas

Este artigo elaborou um método para identificar *browser fingerprinting* empregando a análise da AST de código Javascript. Além disso, demonstrou, através dos dados avaliados, que nunca houve uma queda na coleta de objetos Javascript relacionados a *browser fingerprinting* no decorrer dos anos entre 2016 a 2021. Mostrou que, ao contrário do que se pensava, a coleta de informações através de objetos Javascript está em uma linha constante de crescimento e, por está razão, avalia-se que os sites estão utilizando *browser*

[2]https://drive.google.com/drive/folders/1kPKls6qhLK9eoKssxTF5oYu95WXiy7tR

fingerprinting para coletar mais informações e realizar mais rastreamento do usuário na web.

Este trabalho também demonstrou que a informação do usuário é coletada em cada ação que ele faz em uma página web, pois é grande a quantidade de invocações de chamadas que são encontradas ao se examinar os códigos Javascript das páginas secundárias. Por fim, avançou um passo na fronteira de pesquisa na área de *browser fingerprinting*, pois deu continuidade ao trabalho de [Saraiva et al. 2016], implementando seu método de classificação na **WBF Analyzer** e resolvendo um dos problemas de ofuscação de termos *fingerprinting* citado em sua seção de trabalhos futuros.

7.2. Trabalhos Futuros

Como trabalhos futuros sugere-se: (i) desenvolver uma interface web para a **WBF Analyzer**; (ii) primorar o algoritmo da normalização da **WBF Analyzer** para identificar casos complexos de ofuscação de string de substituição de palavra chave; (iii) aprimorar **WBF Analyzer** para identificar chamadas de objetos Javascript relacionados a *web phishing*.

8. Agradecimentos

Esta pesquisa foi realizada com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001 e, conforme previsto no Art. 48 do decreto nº 6.008/2006, foi parcialmente financiada pela Samsung Electrônica da Amazônia Ltda, nos termos da Lei Federal nº 8.387/1991, através de convênio nº 003/2019, firmado com o ICOMP/UFAM.

Referências

Cooper, A., Tschofenig, H., Aboba, D. B. D., Peterson, J., Morris, J., Hansen, M., and Smith, R. (2013). Privacy Considerations for Internet Protocols. RFC 6973.

Elleres, P., Menezes, A., and Feitosa, E. (2017). Detecção de canvas fingerprinting em páginas web baseada no modelo vetorial. In *Anais do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 456–469, Porto Alegre, RS, Brasil. SBC.

Matos, G. F. d. and Feitosa, E. (2020). Identificando indicadores de browser fingerprinting em páginas web. In *Anais do XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 1–6, Porto Alegre, RS, Brasil. SBC.

Rausch, M., Good, N., and Hoofnagle, C. J. (2014). Searching for indicators of device fingerprinting in the javascript code of popular websites. In *Proceedings, Midewest Instruction and Computing Symposium*.

Saraiva, A., Menezes, A., and Feitosa, E. (2016). Determinando o risco de fingerprinting em páginas web. In *Anais do XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 254–267, Porto Alegre, RS, Brasil. SBC.

Saraiva, A. R., Elleres, P. A. d. P., Carneiro, G. d. B., and Feitosa, E. (2014). Device fingerprinting: Conceitos e técnicas, exemplos e contramedidas. In Santos, A. d., editor, *Minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2014*, pages 49–98. SBC, Porto Alegre.

van Vulpen, B. (2020). Towards finding browser fingerprinters through automated static analysis of javascript code. Bachelor thesis Computing Science. Radboud University.