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

Adriana Rodrigues Saraiva

Determinando o Risco de Fingerprinting em Páginas Web

Manaus
Agosto de 2016

Adriana Rodrigues Saraiva

Determinando o Risco de Fingerprinting em Páginas Web

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

Orientador: Prof. Dr. Eduardo Luzeiro Feitosa

Manaus
Agosto de 2016

Ficha Catalográfica

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

Saraiva, Adriana Rodrigues
S243d Determinando o Risco de Fingerprinting em Páginas Web / Adriana Rodrigues Saraiva. 2016
94 f.: il. color; 31 cm.

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

1. website fingerprinting. 2. ratreamento. 3. risco. 4. privacidade. I. Feitosa, Eduardo Luzeiro II. Universidade Federal do Amazonas III. Título

FOLHA DE APROVAÇÃO

"Determinando o Risco de Fingerprinting em Páginas Web"

ADRIANA RODRIGUES SARAIVA

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

Prof. Eduardo Luzeiro Feitosa - PRESIDENTE

Prof. Arilo Claudio Dias Neto - MEMBRO INTERNO

Prof. Rafael Roque Aschoff - MEMBRO EXTERNO

Manaus, 04 de Agosto de 2016

Aos meus pais, Antônio Maquizanor e Sebastiana Rodrigues e ao meu padastro Ismael Dantas pelo amor, carinho, força e ajuda para conseguir alcançar meus objetivos. Aos meus filhos Adria Caroline e Samuel e esposo Everaldo por entenderem os momentos de ausência.

Agradecimentos

Primeiramente a Deus, pelo dom da vida.

Ao meu orientador, Prof. Eduardo Feitosa, pelo apoio, ensinamentos e orientações acadêmicas que me possibilitaram alcançar essa conquista. Sempre acreditando que eu pudesse ir muito mais além do que eu imaginava e com isso alcançar melhores resultados, pela paciência e confiança que foram fundamentais para conclusão deste trabalho.

Aos meus pais Antônio Maquizanor e Sebastiana Rodrigues pela motivação, força e coragem para sempre seguir em frente com os meus objetivos de vida.

Ao meu padastro Ismael Dantas, pela imprescindível ajuda e apoio.

Aos meus filhos Adria Caroline e Samuel por serem o motivo das minhas conquistas. Ao meu esposo Everaldo pela paciência e ajuda nos momentos de ausência.

Aos meus irmãos Alessandro, Alessandra e Adriano, pela companhia, pela paciência e apoio.

Agradeço também, aos meus amigos Adria, Awdren, Carlos, Edma, Karla Susiane, Isomar, Maria Azevedo, Michel, Pablo Elleres, Rayol, Socorro Brasil e Thaís que em momentos distintos foram primordiais para que eu pudesse ingressar, cursar e concluir este Mestrado. E a todos que, de alguma forma, contribuíram para a concretização desta conquista.

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

Resumo

Técnicas de *fingerprinting* são aquelas empregadas para identificar (ou re-identificar) um usuário ou um dispositivo através de um conjunto de atributos (tamanho da tela do dispositivo, versões de softwares instalados, entre muitos outros) e outras características observáveis durante o processo de comunicação. Comumente conhecidas por *Website fingerprinting*, tais técnicas podem ser usadas como medida de segurança (na autenticação de usuários, por exemplo) e como mecanismo para vendas / marketing. Por outro lado, também podem ser consideradas uma ameaça potencial à privacidade Web dos usuários, uma vez que dados pessoais e sigilosos podem ser capturados e empregados para fins maliciosos nos mais variados tipos de ataque e fraudes. Neste contexto, esta dissertação propõe uma metodologia para detectar artefatos (scripts) *fingerprinting* em páginas Web e mensurar o nível de severidade à privacidade do usuário. Os resultados mostram que embora simples, a metodologia é eficaz ao encontrar códigos *fingerprinting* nos websites e categorizá-los em níveis de severidade.

Palavras-chave: *Website fingerprinting*, rastreamento, risco, privacidade

Abstract

Fingerprinting techniques are those used to identify (or re-identify) a user or device with a set of attributes (device screen size, versions of installed software, among many others) and other observable characteristics during the communication process. Commonly known by Website fingerprinting, such techniques can be used as a security measure (in user authentication, for example) and as a mechanism for sales / marketing. However, they can also be considered a potential threat to Web users' privacy, since personal and sensitive data can be captured and used for malicious purposes in various types of attacks and fraud. In this context, this work proposes a methodology to detect fingerprinting artifacts in Web pages and measure the level of severity to user privacy. The results show that although simple, the method is effective to find fingerprinting codes in websites and categorizing them in severity levels.

Keywords: *Website fingerprinting*, tracking, risk, privacy

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

A Internet se tornou realidade na vida de milhões de pessoas. A busca por informações, comunicação em redes sociais, compras, jogos on-line e até *Internet Banking* são atividades populares e cotidianas. O fato é que os inúmeros avanços tecnológicos proporcionaram aos usuários uma maior acessibilidade, agilidade e dinamismo no acesso às informações e na execução de tarefas. Entretanto, esses mesmos avanços acabaram por permitir a manipulação de uma das características originais da Internet, o anonimato.

Atualmente, o simples ato de navegar deixa uma série de "rastros". Por exemplo, quando um usuário acessa uma página Web, ele tem o seu dispositivo (computador, tablet, smartphone, entre outros) identificado por meio do endereço IP(Internet Protocol). Assim, basta uma consulta a bases de dados, como o Whois[1], e a serviços de geolocalização para descobrir onde o usuário está e de qual local ele está fazendo o acesso.

Outro modo de identificar usuários, e sem dúvida o mais comum, é por meio de *Cookies*[2]. Embora tenham sido criados para personalizar a navegação dos usuários, os *Cookies* passaram a ser utilizados para registrar informações com fins comerciais, em especial com o objetivo de vender informações do usuário (hábitos de navegação e de consumo, por exemplo) a empresas de comércio eletrônico. Isso fez com que a capacidade de rastrear um usuário se tornasse uma atividade altamente lucrativa para empresas especializadas em anúncios.

Empresas de publicidade on-line atuam em parceria com vários sites Web para coletar dados dos usuários durante a navegação e, assim, construir um perfil detalhado dos seus interesses e suas atividades. Mesmo que alguns usuários não vejam problema nisso (que mal existe em receber anúncios on-line relevantes?),

[1]Whois é um protocolo específico para consultar informações de contato e DNS (Domain Name System) sobre entidades na internet

[2]Cookies são pequenos pedaços de texto (arquivos) que os sites Web fazem o navegador do usuário salvar com certos objetivos

a posse de dados de usuários, coletados sem consentimento por empresas de publicidade on-line, ou mesmo por qualquer empresa ou agência governamental, traz consequências potencialmente desastrosas para a privacidade das pessoas e, em casos mais graves, pode envolvê-las em roubos, fraudes e ataques maliciosos.

1.1 Motivação

Devido ao fato do uso de *Cookies* ser visto como um grave problema, em 1997 a RFC (Request For Comments) 2109 [1] foi proposta limitando seu uso, especialmente aqueles que envolvem terceiros como empresas de publicidade. Na época, os dois navegadores dominantes do mercado, *Netscape Navigator* e *Internet Explorer*, se comprometeram a adotar formalmente a especificação, mas na prática nada aconteceu e os *Cookies* se tornaram peça fundamental na identificação e rastreamento de usuários. No entanto, em 2005, os desenvolvedores de navegadores começaram a adicionar o modo de navegação privada em seus produtos, visando dar aos usuários a opção de visitar sites Web sem deixar *Cookies* de longo prazo. Seguindo essa "tendência", desenvolvedores independentes iniciaram a produção de extensões para preservar a privacidade dos usuários. *AdBlockPlus*[3], *Ghostery*[4] e *Lightbeam*[5] são bons exemplos desse tipo de solução.

Entretanto, a indústria de anúncios e publicidade, interessada em manter sua continuação nesse "ramo de atividade", não ficou parada. Aproveitando-se do surgimento e proliferação dos dispositivos móveis, bem como das novas possibilidades de comunicação, essas empresas modificaram a forma de rastreamento dos usuários. Agora, não é mais necessário usar servidores Web que deixam rastros (migalhas) na máquina do usuário. Ao invés disso, resolveram apostar em ferramentas de reconhecimento/rastreio por meio do próprio conjunto de hardware/software do usuário.

Esse conceito, conhecido como *Website Fingerprinting* [2], parte do pressuposto que cada usuário opera o seu próprio hardware. Assim, a identificação de um dispositivo é o mesmo que identificar a pessoa por trás dele. Segundo Nikiforakis [2], *Website Fingerprinting* faz uso de um conjunto de atributos do sistema extraídos do dispositivo do usuário, que geram uma combinação de valores únicos capazes de identificar um usuário/dispositivo.

O fato relevante e motivador para esta pesquisa é que as técnicas de *Website Fingerprinting* estão se tornando mais comuns e que os usuários sabem pouco ou quase nada sobre o assunto. Mesmo quando estão cientes de que estão sendo monitorados (como uma medida de proteção contra a fraude, por exemplo), eles

[3] https://adblockplus.org
[4] https://www.ghostery.com/
[5] https://addons.mozilla.org/pt-br/firefox/addon/lightbeam/

simplesmente confiam que as informações coletadas não serão utilizadas para outros fins.

1.2 Objetivo

O objetivo desta dissertação é propor uma abordagem para mensurar a severidade de risco de um artefato *fingerprint*, através da avaliação dos elementos (métodos, propriedades, objetos e atributos) presente em uma página Web, a fim de alertar os usuários sobre sua existência e mostrando a noção sobre o perigo ao qual estão sendo expostos.

Especificamente, pretende-se:

- Analisar os elementos (métodos, propriedades, objetos e atributos) de acordo com sua capacidade de gerar ataques ou ameaças à privacidade dos usuários;
- Classificá-los em níveis de severidade de risco;
- Implementar uma prova de conceito, avaliando em diferentes bases de dados de artefatos *fingerprint* (sites Web) a fim de provar a efetividade da abordagem proposta.

1.3 Contribuições Esperadas

Esta dissertação pretende contribuir com a criação de uma abordagem para medir a severidade (risco de ameaça à privacidade) de artefatos fingerprint em páginas Web.

1.4 Estrutura do Documento

Para alcançar os objetivos propostos, o trabalho está organizado da seguinte forma:

O Capítulo 2 apresenta os conceitos, definições e classificações sobre *Website fingerprinting*, bem como uma discussão sobre os problemas de privacidade. Além disso, uma breve explicação sobre os navegadores a fim de demonstrar como é possível identificar um navegador por meio de suas propriedades e métodos. O Capítulo 3 aborda as tecnologias empregadas (ou melhor dizendo, alvo) em um *Website fingerprinting*.

Os trabalhos relacionados são discutidos no Capítulo 4. O objetivo deste capítulo é apresentar trabalhos relacionados ao tema em estudo e as possíveis soluções existentes que focam na detecção de *Website fingerprinting*.

O Capítulo 5 apresenta a proposta da dissertação e sua implementação, na qual destaca-se de maneira detalhada uma visão geral da abordagem proposta para detecção de *Website Fingerprinting*, assim como, os passos para sua implementação.

No Capítulo 6 são descritos, em duas seções, o ambiente de experimentação e resultados obtidos com a aplicação da abordagem proposta.

Por fim, o Capítulo 7 apresenta as considerações finais e dificuldades encontradas no decorrer da pesquisa, assim como, a identificação de trabalhos futuros.

Capítulo 2

Conceitos Básicos

Este Capítulo está dividido em três partes. Primeiramente, *Website fingerprinting* é definido, classificado, e o problema de privacidade causado por ele é discutido. Em seguida, os principais pontos que relacionam o navegador com *Website fingerprinting* são levantados e por fim, a metodologia OWASP (Open Web Application Security Project) é descrita, na qual a classificação dos *Website fingerprinting* é baseada.

2.1 *Website Fingerprinting*

O termo *fingerprinting* ganhou força na área da computação nos anos de 1990 com o surgimento de várias ferramentas especializadas em realizar ataques a redes de computadores. Isso porque percebeu-se que para efetuar um ataque bem sucedido era necessário descobrir/identificar corretamente a máquina (computador/servidor) alvo, o sistema operacional que ela executava e os aplicativos ativos. Ferramentas como o Nmap[1] surgiram nessa época e são bastante utilizadas até hoje. No âmbito Web, foco desta dissertação, o termo *fingerprinting* veio à tona em 2009 quando Mayer [3] observou que as características de um navegador e seus plugins podiam ser identificadas e os usuários rastreados.

Formalmente, o termo *fingerprint* foi definido na RFC 6973 [4] 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".

Esta dissertação compartilha a visão de que *fingerprinting* é parte de um conjunto amplo de tecnologias e técnicas, também conhecidas como *Device in-*

[1] http://nmap.org

telligence, *Machine fingerprinting*, *Browser fingerprinting*, *Web fingerprinting* ou *Device Fingerprinting*, usadas para identificar (ou re-identificar) um usuário ou um dispositivo por meio de um conjunto de configurações, atributos (tamanho da tela do dispositivo, versões de software instalado, entre muitos outros) e outras características observáveis durante comunicações. Nesta dissertação, os termos *Device fingerprinting* e *fingerprinting* serão usados para representar essas técnicas de identificação com foco na Web.

Dois aspectos interessantes e relevantes podem ser observados sobre *fingerprinting*. O primeiro é que embora tenham sido popularizadas no ambiente Web, tais técnicas estão presentes no mundo móvel. Trabalhos como [5] e [6] propõem diferentes soluções, métodos e técnicas para identificar um usuário baseado em conjuntos únicos de atributos como: números discados, tempo das ligações, conexão com a estação rádio base, entre outras. Segundo, tais técnicas tem um grande potencial de ameaça à privacidade dos usuários na Web. Questões relacionadas aos problemas de privacidade serão discutidas na Seção 2.1.2. É importante, também, ressaltar que trabalhos recentes apontam o uso dessas técnicas para engendrar ataques (persistentes e direcionados) [7, 8].

2.1.1 Classificação

De acordo com o W3C (*World Wide Web Consortium*)[2] [9], as técnicas de *Website fingerprinting* podem ser classificadas em três tipos:

1. **Passiva**: Também chamada de *Browser fingerprinting*, é aquela baseada nas características observáveis no conteúdo de solicitações Web, sem a utilização de qualquer código em execução no lado do cliente. Esse tipo de *fingerprinting* inclui o conjunto de cabeçalhos de solicitação HTTP (Hypertext Transfer Protocol), endereço IP e outras informações do nível de rede. Eventualmente, *Cookies* também são utilizados.

2. **Ativa**: Levam em consideração técnicas onde o site Web é executado, via JavaScript (ou outro código), no lado do cliente para observar características adicionais sobre o navegador. Técnicas para *fingerprinting* ativas podem incluir o acesso ao tamanho da janela, enumerar fontes ou plugins, avaliação das características de desempenho ou os padrões de renderização de gráficos.

3. **Cookie-like**: Nessa categoria, usuários, *user-agents*[3] e dispositivos também podem ser re-identificados por um site que primeiro configura e de-

[2] http://www.w3c.org

[3] *user-agent* é um componente do cabeçalho do protocolo HTTP.

pois recupera o estado armazenado de um navegador ou dispositivo. A re-identificação de um usuário ou inferências sobre ele, da mesma forma que os *Cookies* permitem o gerenciamento de estado para o protocolo HTTP (RFC6265 [10]). Também podem contornar as tentativas do usuário em limitar ou apagar os *Cookies* armazenados pelo *user-agent*, como demonstrado em [11].

2.1.2 Privacidade do Usuário

Embora grande parte do desenvolvimento das técnicas de *Website fingerprinting* esteja relacionada à identificação de usuários como medida de segurança e mecanismo anti-fraude (empresas como BlueCava[4], Iovation[5] e ThreatMetrix[6] são bons exemplos dessa finalidade), os impactos na privacidade dos usuários causados por essas técnicas são ameaças reais de segurança. Três problemas na privacidade são apontados em [9]:

1. **Identificação do usuário**: Até por questões de padronização, a maioria dos usuários prefere ficar anônimo quando navegam na Internet. Segurança física e pessoal, discriminação, sigilo de dados, entre outros, são motivos para os usuários ficarem no anonimato. No entanto, um *fingerprinting* tem o potencial de obter esses dados, sem autorização prévia, deixando o usuário exposto a todos esses fatores.

2. **Correlacionar as atividades da navegação**: Problemas com a privacidade ocorrem, mesmo que o usuário não seja identificado. Os usuários podem se surpreender ao perceber que terceiros (empresas de publicidade on-line em sua maioria) podem correlacionar suas várias visitas ao mesmo ou diferentes sites para elaborar um perfil do usuário. A preocupação aumenta uma vez que essa invasão de privacidade pode acontecer com ou sem autorização do usuário, sem mencionar que ferramentas, como as que fazem a limpeza de *Cookies*, não impedem essa correlação.

3. **Inferências sobre o usuário**: As informações coletadas durante um *fingerprinting* podem revelar dados sobre os quais se pode tirar conclusões sobre o usuário. A versão do sistema operacional e informações sobre a CPU são exemplos de dados que podem ser usados para inferir, por exemplo, o poder de compra do usuário. O W3C [9] afirma que tais inferências permitem oferecer e mostrar ao usuário produtos em determinada faixa de

[4] http://www.bluecava.com
[5] http://www.iovation.com
[6] http://www.threatmetrix.com

preço, o que pode ser uma forma discriminatória de publicidade. Sem falar que os usuários podem se sentir tratados de forma diferente.

Um exemplo de como o *fingerprinting* é capaz de monitorar e correlacionar atividades de navegação de um usuário, dentro e através de sessões, e coletar informações que podem inferir sobre suas preferências e hábitos é apresentado na Figura 2.1.

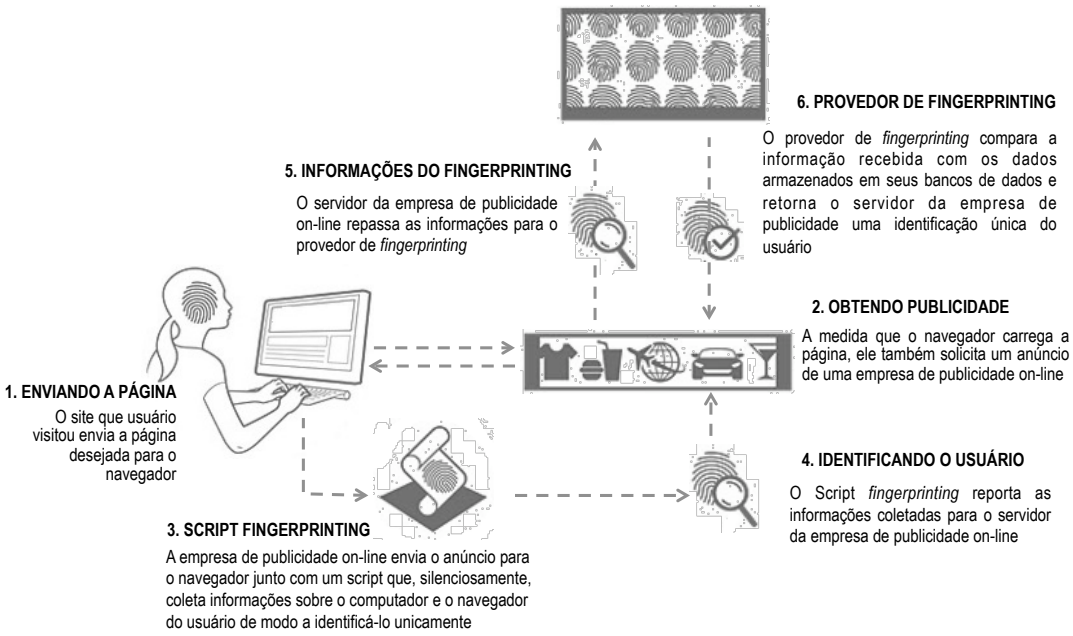

Figura 2.1: Exemplo do processo de *fingerprinting*. Fonte: [2]

Inicialmente, ao acessar uma página de comércio eletrônico (1), o usuário vê, automaticamente, os anúncios de publicidades serem carregados (2). O que ele não sabe é que junto vão scripts de *fingerprinting* com o objetivo de coletar suas informações sem prévia autorização (3). Assim que as informações são coletadas, as mesmas são enviadas para o servidor da empresa de publicidade (4), que as repassa para o provedor *fingerprinting* (5) - que pode ser a mesma empresa. Após receber todos os dados, o provedor compara as informações recebidas com os dados armazenados em sua base a fim de gerar uma identificação única para este usuário ou apenas revalidá-lo, sendo tal informação enviada a empresa de publicidade (6).

2.1.3 Usos de *Fingerprinting*

Embora, o processo de *fingerprinting* possa parecer direcionado para atividades maliciosas, isso não é de todo verdade. Grandes empresas, como ThreatMetrix, Iovation e Bluecava, vendem soluções de *fingerprinting* para evitar fraudes online [12]. Além disso, *fingerprinting* já é bastante utilizado para melhorar a experiência do usuário ao acessar conteúdo Web. Outros exemplos de uso para o bem são: a prevenção contra sequestro de contas [13], serviço anti-bots, remoção de cookies, recomendações de serviços e/ou produtos, entre outros.

Neste trabalho, o foco é provar a existência de *fingerprinting* e classificar o risco ao qual o usuário está exposto. Vale ressaltar que embora exista o elemento (termo) associado à *fingerprint*, não é possível afirmar que sua presença significa um ataque. Maiores detalhes sobre os níveis de risco são apresentados no Capítulo 5.

2.2 Navegador

Nos dias de hoje, é impossível negar e contestar a presença dos navegadores Web na rotina dos usuários em alguns sistemas de computação. O motivo é bastante simples. As aplicações Web tornaram-se tão populares a ponto de rivalizar com software e aplicações nativas. Neste contexto, os navegadores tornaram-se a interface dominante (mecanismo padrão) de conexão dos usuários com sistemas e aplicações Web, bem como conteúdos e serviços de seu interesse.

Três fatos comprovam a ascensão dos navegadores. O primeiro é a grande movimentação por parte das fabricantes de Sistemas Operacionais em desenvolverem seus próprios navegadores [14]. A Microsoft tem o Internet Explorer, a Apple criou o Safari, a Google criou um sistema operacional, o ChromeOS, baseado inteiramente no navegador Chrome, e a Mozilla criou o FirefoxOS.

O segundo é que os fabricantes de navegadores disponibilizam novas versões com uma rapidez impressionante, alguns com intervalos de tempo bem reduzidos e sempre trazendo ou agregando novas funcionalidades, especialmente nas versões voltadas aos dispositivos móveis. O terceiro e último é a grande competitividade no mercado mundial de navegadores.

Recente pesquisa da StatCounter [15] aponta o navegador Chrome como o mais utilizado entre os internautas no período de Janeiro a Julho de 2016, englobando 58,26% da preferência dos usuários. Em seguida vem o Firefox com 13,97%, o Internet Explorer (IE) com 9,77%, o Safari com 9,74% e o Opera com 1,77%.

No que diz respeito ao *Website fingerprinting*, o problema com os navegadores é a não adoção dos padrões. Toda a interação e a forma como o navegador

interpreta e exibe os recursos solicitados pelo usuário segue padrões definidos e mantidos pelo W3C . Contudo, os fabricantes de navegadores mantiveram-se, por muito tempo, parcialmente de acordo com essas especificações, o que lhes permitiu desenvolver suas próprias extensões e com isso agregar novas funcionalidades. Embora hoje a maioria dos navegadores estejam relativamente de acordo com as especificações, essa diferença nos padrões causa sérios problemas de compatibilidade. Além da questão estética, a não completa adoção dos padrões traz implicações relevantes na segurança e privacidade dos usuários e informações. É exatamente neste ponto que entram ou se encaixam as técnicas de *Website fingerprinting*.

2.2.1 Arquitetura do Navegador

A arquitetura de um navegador une diversas funcionalidades de forma a tornar capaz de apresentar os sites Web. Contudo, alguns itens são imprescindíveis e fazem parte de sua arquitetura. Os principais componentes de um navegador utilizados por técnicas de *Website fingerprinting* são: Motor de Navegação, Motor de Renderização, Motor de Interpretação de JavaScript.

1. **O Motor de Navegação** (*Browser Engine*) é o encarregado pela comunicação das entradas da interface do usuário em conjunto com o motor de renderização (*Rendering Engine*). Seu papel é consultar e manipular o motor de renderização, de acordo com as requisições que ocorrem entre a aplicação e a interface de usuário.

2. **O Motor de Renderização** é o componente responsável por exibir o conteúdo solicitado na tela do navegador. Também chamado de motor de layout, tem, por padrão, a função de exibir documentos HTML (HyperText Markup Language), XML (eXtensible Markup Language) e imagens. Pode também exibir outros tipos de dados por meio de plugins ou extensões, isto é, para exibir documentos PDF é preciso um plugin visualizador de PDF. Os navegadores usam diferentes motores de renderização. O Internet Explorer usa Trident, o Firefox usa Gecko, o Safari usa WebKit. O Chrome (a partir da versão 28) e o Opera (a partir da versão 15) usam Blink, uma ramificação do WebKit. Por fim, o motor de renderização está interligado com execuções do interpretador de JavaScript em processos que ocorrem em tempo de execução.

3. **O Motor de Interpretação de JavaScript** é usado para interpretar e executar códigos JavaScript.

Para melhor explicar os elementos empregados em *Website Fingerprinting*, é preciso entender como os conteúdos de um página Web são estruturados. Em outras palavras, como DOM (*Document Object Model*) funciona.

2.2.2 DOM - Document Object Model

A W3C [16] define DOM como sendo uma plataforma de interface que possui uma linguagem neutra, cuja finalidade é permitir que programas e scripts possam acessar e atualizar o conteúdo, a estrutura e o estilo dos documentos de forma dinâmica. O nós de cada documento são organizados em uma estrutura de árvore, chamada de árvore DOM.

Os objetos na árvore DOM podem ser endereçados e manipulados pelo uso de métodos sobre os objetos. Atualmente alguns desses objetos estão sendo usados para composição de um *Website Fingerprinting*, inseridos em scripts executados pelos websites.

É importante destacar que entender o DOM permite ter uma visão geral sobre como ocorre o processo de execução dinâmica no navegador e entender o papel das linguagens scripts nesse processo, oferecendo flexibilidade necessária à inclusão, alteração ou exclusão de conteúdos ou nós da estrutura DOM que serão processados, executados e exibidos no navegador.

A partir do uso de tecnologias Web, observa-se frequentemente a exploração maliciosa desses recursos, por meio de pontos de ataques no código HTML, CSS (Cascading Style Sheets) e principalmente em linguagens *scripts*, sendo que a linguagem JavaScript apresenta-se como principal vetor.

A Figura 2.2 ilustra os objetos e elementos da estrutura DOM, alguns desses objetos que foram alvo desta pesquisa, podem ser vistos no Apêndice B.

O primeiro objeto na hierarquia DOM manipulado em uma página Web é o **Window**, cuja função é a manipulação das janelas do navegador. Por meio do objeto *Window*, é possível obter acesso aos demais objetos que podem ser usados para compor um artefato *fingerprint*. No apêndice B a Tabela B.1 ilustra as mais importantes propriedades do objeto *Window* aplicadas em *Website Fingerprinting*.

Outro objeto do topo da estrutura DOM é o **Document**, que representa o documento HTML, XML ou SVG (*Scalable Vector Graphics*) que será carregado em uma janela e que mantém o conteúdo da página Web. O objeto *Document* pode ser usado para *fingerprinting*, pois controla todos os elementos contidos na página.

A Tabela B.2 ilustra as mais importantes propriedades do objeto *Document* aplicadas em *Website Fingerprinting*.

Já o objeto **Navigator** possui propriedades que representam informações sobre o navegador como nome, versão, linguagem, entre outras informações. A

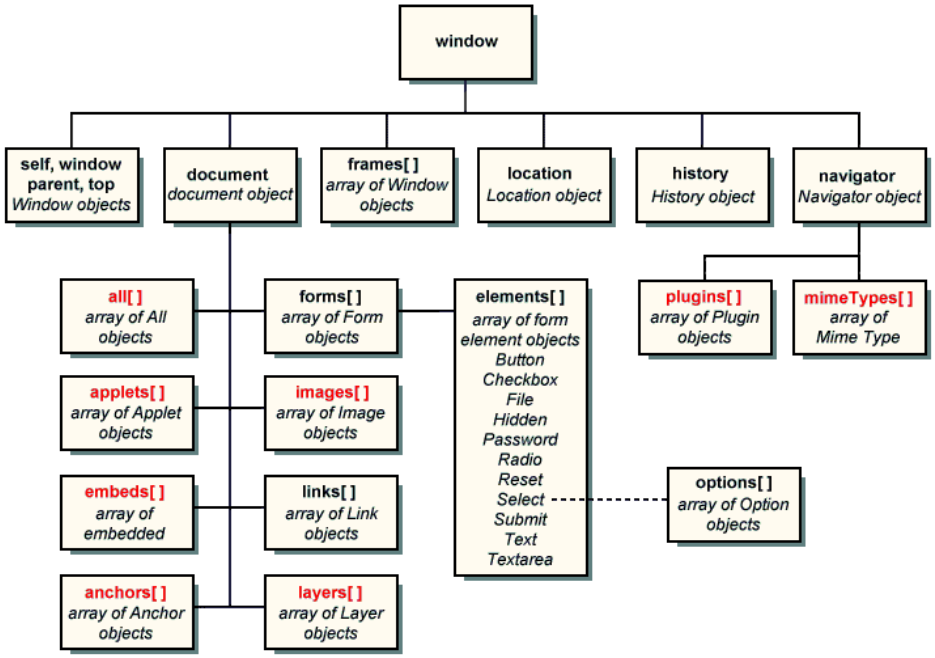

Figura 2.2: Estrutura DOM Nível 0. Fonte:[16]

Tabela B.3 ilustra as mais importantes propriedades do objeto *Navigator* aplicadas em *Website Fingerprinting*.

O objeto **Screen** tem por finalidade obter informações referentes as dimensões de tela do usuário. Segundo Acar [17], estas propriedades podem fazer parte da composição de um *fingerprinting* identificando, por exemplo, se o dispositivo que o usuário acessa a internet, distinguindo entre um computador desktop ou smartphone. A Tabela B.4 ilustra as mais importantes propriedades do objeto *Screen* aplicadas em *Website Fingerprinting*.

2.2.3 Como identificar o Navegador?

Existem várias formas de se identificar um navegador. Esta seção aborda apenas duas das mais simples técnicas capazes desse feito.

User-agent

Para descobrir informações sobre o navegador (e sobre a máquina que o hos-

peda), a ideia mais simples é pedir ao próprio navegador para revelar "voluntariamente' sua verdadeira identidade. O mecanismo mais simples para isso é o *user-agent*, um componente do cabeçalho do protocolo HTTP.

De acordo com a RFC 2616 [18], *user-agent* é um campo enviado quando o navegador faz a solicitação de uma página para: (i) fins estatísticos; (ii) descobrir violações no protocolo; (iii) reconhecimento automatizado do *user-agent* para evitar limitações específicas e melhor exibir o conteúdo solicitado.

O *user-agent* é uma sequência de caracteres composta por um conjunto de elementos, chamados *tokens*, listados em ordem de importância de acordo com o padrão. Por exemplo, **Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.78.2 Version/7.0.6 Safari/537.78.2** é uma resposta típica da execução do *user-agent* de um navegador detalhada na Tabela 2.1.

Tabela 2.1: Campos de um *user-agent*

Token	**Descrição**
Nome da Aplicação	Mozilla
Versão	5.0
Plataforma	Macintosh
Sistema Operacional	Intel Mac OS X 10_9_4 (Versão 10_9_4)
Motor de Layout	AppleWebKit (build 537.78.2)
Versão do Navegador	7.0.6
Nome do Navegador	Safari (build 537.78.2)

Como forma de proteção, nem sempre muito eficaz, os navegadores utilizam extensões que permitem alterar (falsificar) determinadas configurações enviadas pelo *user-agent*, uma vez que é uma sequência string. Para o Internet Explorer existe o plugin *UAPick*[7]. Para o Firefox, o *User Agent Switcher*[8] é o plugin mais usado. A mesma extensão é usada no Chrome[9]. O Safari possui o *User Agent Browser*[10]. No Opera, o *User Agent Changer* é a extensão que permite manipular o *user-agent*.

Na prática, *user-agent* é mais utilizado através do objeto *Navigator*.

Objeto *Nagivator*

[7] http://www.enhanceie.com/ietoys/uapick.asp
[8] https://addons.mozilla.org/pt-BR/firefox/addon/user-agent-switcher/
[9] http://goo.gl/YybG4
[10] https://itunes.apple.com/br/app/user-agent-browser-simply/id414229165?mt=8

Outra forma de identificar um navegador é utilizar o objeto *Navigator*. Para exibir o conteúdo de uma página, o navegador cria automaticamente uma hierarquia de objetos DOM refletindo alguns elementos inseridos na página. Existem três objetos básicos: *Screen*, *Window* e *Navigator*. Este último é o que representa o próprio navegador e através dele é possível controlar seu comportamento, além de obter informações sobre suas características. A especificação do HTML5 [19] diz que qualquer informação sobre o navegador é feita através do atributo *navigator* da interface *Window*, que deve retornar uma instância da interface *Navigator*.

As propriedades do objeto *Navigator* são exibidas na Tabela 2.2. É importante ressaltar que todas essas propriedades são somente de leitura (*ready-only*).

Tabela 2.2: Propriedades do objeto *Navigator*

Propriedades	**Descrição**
appCodeName	Retorna o codinome do navegador.
appName	Retorna uma string DOM com o nome real do navegador
appVersion	Retorna uma string DOM com a versão do navegador
CookieEnabled	Determina se os Cookies estão habilitados no navegador
language	Retorna o idioma do navegador
on-line	Determina se o navegador está on-line
product	Retorna o nome do motor do navegador
platform	Retorna para qual plataforma o navegador foi compilado
user-agent	Retorna o cabeçalho do *user-agent* enviado pelo navegador para o servidor

Na prática, o objeto *navigator* contém a mesma sequência de informações de um *user-agent*. No entanto, ambos não apresentam boas características de segurança, uma vez que podem ser ajustados pelo usuário de forma arbitrária.

2.3 Metodologia de Risco da OWASP

A Metodologia de Risco da OWASP[11] emprega o impacto de um evento e a probabilidade de uma vulnerabilidade ser explorada nesse evento para estimar o risco de um incidente. Contudo, nesta dissertação, uma junção entre impacto e probabilidade foi realizada para estimar a severidade de um artefato *fingerprint*.

[11] é uma entidade sem fins lucrativos e de reconhecimento internacional que contribui para a melhoria da segurança de softwares aplicativos (https://www.owasp.org).

Explicando melhor, na metodologia da OWASP, o **impacto** de um evento é estimado empregando-se dois grupos de fatores: técnico e negócios. O primeiro é alinhado com os pilares tradicionais da área de segurança: confidencialidade, integridade, disponibilidade e auditoria (*accounting*). O segundo avalia, por exemplo, danos financeiros, danos à imagem, falhas no cumprimento/realização de atividades e a violação de privacidade, exigindo uma profunda compreensão do negócio.

Para estimar a **probabilidade** de uma vulnerabilidade ser descoberta e explorada por um atacante, a OWASP faz uso de dois conjuntos de fatores. O primeiro está relacionado com o atacante envolvido e seu objetivo, ou seja, visa estimar a probabilidade de um ataque ter êxito a partir de um grupo de possíveis atacantes. Nível de habilidade, motivação, oportunidade e tamanho são os fatores deste grupo. O segundo conjunto está relacionado com a vulnerabilidade envolvida e seu objetivo é estimar a probabilidade de uma vulnerabilidade ser descoberta e explorada. Facilidade de descoberta, facilidade de exploração, conhecimento e detecção são os fatores deste grupo.

Nesta dissertação, os impactos técnico e nos negócios não são considerados, uma vez que, no máximo, o pilar corrompido é o da confidencialidade e o foco é violação de privacidade. Em outras palavras, este trabalho não executa os sites Web para avaliar os caminhos e consequências na sua execução.

Capítulo 3

Tecnologias empregadas

Este Capítulo descreve as principais tecnologias presentes nos navegadores modernos aptas a atuarem em *Website Fingerprinting*. HTML5, Canvas, JavaScript, Flash Player, WebGL (*Web Graphics Library*), CSS e Silverlight serão explicadas, exemplos de código empregados em *Fingerprinting* serão apresentados e alguns trabalhos de execução e deteção de *Website Fingerprinting* também serão discutidos.

3.1 HTML5

A HTML5 é a quinta versão do padrão HTML, que desde 2014 é recomendada pela W3C. Assim como suas predecessoras, a principal finalidade da HTML5 é estruturar o conteúdo que se apresenta na Web, através do suporte as mais recentes tecnologias multimídia. Este novo padrão engloba não só a HTML, mas também a XHTML1 (eXtensible Hypertext Markup Language) [20] e o HTML DOM Nível 2 [21]. Por isso é vista como uma tentativa de definir uma única linguagem de marcação que pode ser escrita em qualquer uma das sintaxes mencionadas anteriormente.

O padrão HTML5 define novos elementos, atributos e comportamentos, bem como possui suporte a um conjunto bem maior de tecnologias, o que permite o desenvolvimento de aplicações Web e sites mais dinâmicos e poderosos. HTML5 também é um candidato em potencial para aplicações móveis multiplataforma. Muitos recursos foram construídos com o requisito de serem capazes de executar em dispositivos de baixa potência, como smartphones e tablets.

Embora as especificações da HTML5 tenham sido finalizadas recentemente, os principais navegadores, especialmente o Mozilla e o Google Chrome, começaram a implementar partes deste padrão logo após o surgimento das primeiras especificações. Tal fato, em conjunto com várias funções experimentais, trouxe-

ram problemas como *fingerprinting*. Por exemplo, HTML5 permite verificar se o navegador suporta a API (Application Programming Interface) de geolocalização (o que permite usar a propriedade de geolocalização no objeto *Navigator*) e permite que aplicações Web armazenem informações localmente dentro do navegado do usuário. A listagem 3.1 ilustra dois scripts (JavaScript) que executam os exemplos descritos.

Listagem 3.1: Dois exemplos de características HMTL5 usadas em *Website Fingerprinting*

```
1  <script>
2  var x = document.getElementById("demo");
3  function getLocation() {
4      if (navigator.geolocation) {
5          navigator.geolocation.getCurrentPosition(showPosition);
6      } else {
7          x.innerHTML = "Geolocation is not supported by this browser.";
8      }
9  }
10 function showPosition(position) {
11     x.innerHTML = "Latitude: " + position.coords.latitude +
12     "<br>Longitude: " + position.coords.longitude;
13 }
14 </script>
15
16 ....
17
18 <script>
19 // Check browser support
20 if (typeof(Storage) != "undefined") {
21     // Store
22     localStorage.setItem("lastname", "Smith");
23     // Retrieve
24     document.getElementById("result").innerHTML = localStorage.getItem("
           lastname");
25 } else {
26     document.getElementById("result").innerHTML = "Sorry, your browser does
           not support Web Storage...";
27 }
28 </script>
```

O primeiro trecho de código (linhas 1 até 18) usa a API de Geolocalização do HTML para checar se o navegador suporta essa característica. Se sim, o método *getCurrentPosition()* é executado, retornando um objeto que é usado como parâmetro para a função *showPosition*, responsável pode exibir a latitude e longitude do usuário.

O segundo código (linhas 19 a 28) permite armazenar dados sem data de expiração. Para tanto, verifica se o armazenamento local é suportado. Se sim, um atributo *localStorage* é criado contendo os valores $name = "lastname"$ e $value = "Smith"$ e, em seguida, o valor "lastname"é armazenado localmente. Vale ressaltar que o armazenamento local é mais seguro e permite até 5Mbytes por arquivo, sem afetar o desempenho do site Web e nunca enviando os dados armazenados para o servidor.

3.2 HTML5 Canvas

Canvas é um novo elemento da HTML5 que fornece uma área da tela que pode ser utilizada via programação. Através de JavaScript, Canvas permite o acesso a um conjunto completo de funções de desenho, permitindo que gráficos sejam gerados dinamicamente. Os autores Fulton et al. [22] destacam que Canvas é o equivalente a uma lona utilizada por artistas como superfície de pintura. Em termos práticos, Canvas é uma área bitmap que pode ser manipulada pelo JavaScript.

Dentre os principais recursos que o Canvas disponibiliza, pode-se destacar: (i) **Interatividade**: Canvas pode responder às ações do usuário, através dos eventos de teclado, o mouse ou toque; (ii) **Animação**: Cada objeto desenhado na tela pode ser animado; (iii) **Flexibilidade**: É possível que os desenvolvedores criem qualquer coisa; (iv) **Padrão Web**: Canvas é uma tecnologia aberta que faz parte da HTML5; e (v) **Portabilidade**: Ao contrário do Flash e Silverlight, uma aplicação com Canvas pode ser executada em praticamente qualquer lugar.

Entretanto, Canvas faz parte das tecnologias modernas aptas a atuar em *Website Fingerprinting*. A listagem 3.2 ilustra uma simples prova de conceito [23], em JavaScript, que possibilita executar um *fingerprinting*. O método é baseado no fato de que um mesmo elemento HTML5 Canvas pode produzir pixels excepcionais em diferentes navegadores, dependendo do sistema no qual é executado. Isso acontece por duas razões. A primeira é o formato de imagem. Navegadores utilizam diferentes mecanismos de processamento de imagem, opções de exportação, níveis de compressão, o que faz com que as imagens finais possam receber diferentes *hashes*. A segunda é no nível de *pixmap*, uma vez que sistemas operacionais usam diferentes algoritmos e definições de *anti-aliasing*[1] e renderização de pixels.

Listagem 3.2: Código exemplo de HTML5 Canvas

```
<script type="text/javascript">
    var Canvas = document.getElementById("Canvas");
    var context = Canvas.getContext("2d");
    context.fillStyle = "rgb(255, 0, 0)";
    context.fillRect(30, 30, 50, 50);
    context.font = "20px serif";
    context.fillStyle = "rgb(0, 0, 255)";
    context.fillText("Hello World", 100, 100);
</script>
```

O resultado da listagem 3.2 é um quadrado vermelho e um texto (Figure 3.1).

Tornando o cenário um pouco mais preocupante, o trabalho de Mowery e Shacham [24] observou que é possível relacionar o navegador, com maior intimidade,

[1]

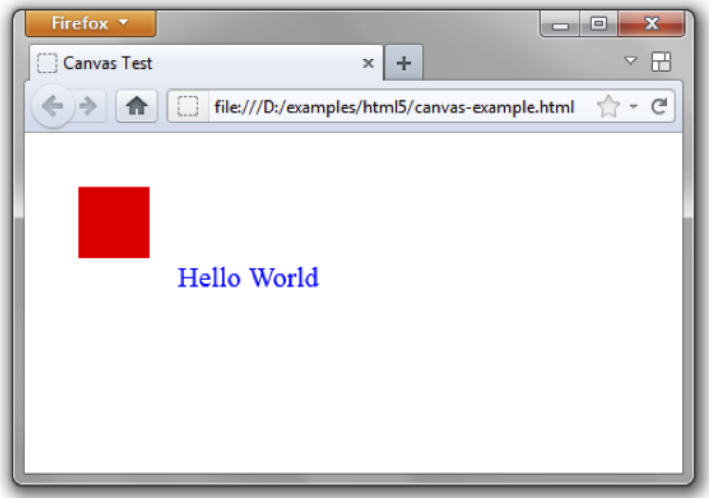

Figura 3.1: Exemplo de HTML5 Canvas. Fonte: [23]

às funcionalidades do hardware e do sistema operacional. Os autores desenvolveram uma técnica de *fingerprinting* onde quando um usuário visita um site que utiliza a técnica, o navegador é instruído a desenhar uma linha oculta de texto ou de gráfico 3D, que é então convertida em um sinal digital. Desta maneira, as variações nas quais a GPU (Graphics Processing Unit) está instalada ou o driver gráfico causam variações em *tokens* digitais. O *token* pode ser armazenado e compartilhado com empresas de publicidade para identificar os usuários quando eles visitam sites afiliados. Um perfil pode ser criado como atividade de navegação de um usuário, permitindo que anunciantes direcionem sua publicidade de acordo com as preferências do usuário.

O estudo de Kirk [25] relata uma pesquisa que encontrou um código utilizado para *fingerprinting*, usando Canvas, que estava em uso no início deste ano em mais ou menos 5000 sites populares, sem qualquer tipo de conhecimento para seus usuários. Entretanto, nem todos os locais observados com *fingerprinting* faziam o compartilhamento de conteúdo para empresas de publicidade. Ele ressalta ainda que as empresas europeias estão à procura de novas maneiras de entregar publicidade segmentada aos usuários, afastando-se dos Cookies.

Um exemplo ilustrativo acerca desta técnica de *fingerprinting* é apresentado na Figura 3.2, onde o fluxo de operações do *fingerprinting* com Canvas é descrito em 3 passos. De modo geral, quando um usuário visita uma página, o script *fingerprinting* primeiro desenha textos com a fonte e o tamanho de sua escolha e acrescenta cores de fundo (1). Em seguida, o script chama o método **ToDataURL**, da API Canvas, para obter os dados de pixel da tela em formato **DataURL** (2), que é basicamente uma representação codificada em Base 64 dos

dados de pixel binários. Por fim, o script leva o *hash* dos dados de pixel codificada de texto (3), que serve como *fingerprint* e pode ser combinada com outras propriedades de alta entropia do navegador, como a lista de plugins, a lista de fontes ou a string *user-agent*.

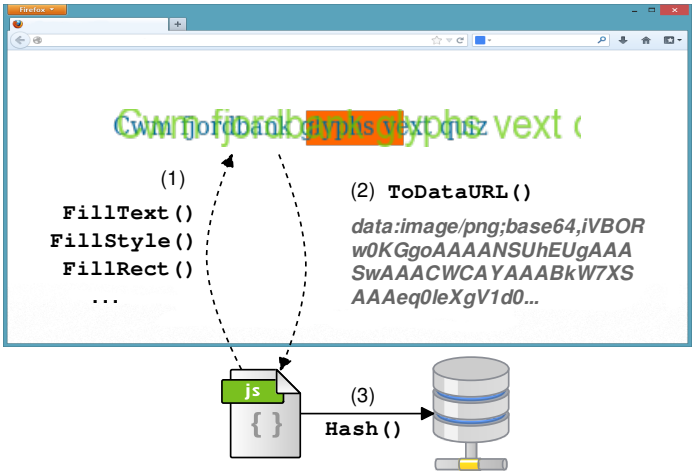

Figura 3.2: Exemplo de *Fingerprinting* utilizando Canvas. Fonte: [26]

3.3 JavaScript

O JavaScript é uma linguagem de programação interpretada, implementada como parte dos navegadores para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade deste script passar pelo servidor [27]. Desenvolvida pela Netscape Communications Corp., por Brendan Eich, possui uma sintaxe similar ao Java e ao C++ e é orientada a objetos, o que permite tratar todos os elementos da página como objetos distintos, facilitando a tarefa da programação.

Entre os argumentos para empregar JavaScript em *Website Fingerprinting* pode-se mencionar: (i) JavaScript é uma tecnologia bem estabelecida, padronizada como ECMAScript [28]; (ii) é suportada por todos os principais navegadores; (iii) funciona em dispositivos móveis; (iv) é utilizada por quase todos os sites; e (v) é ativado por padrão em todos os navegadores. Em comparação com outras abordagens de identificação, o uso de JavaScript é mais robusto e não facilmente transportado de um navegador para outro.

A listagem 3.3 ilustra um simples exemplo de como JavaScript pode ser usado para *Website Fingerprinting*. O código a seguir lista todos os plugins instalados no navegador.

Listagem 3.3: Código para listar os plugins instalados no navegador

```
var plugins = (function(){
    var found = {};
    var version_reg = /[0-9]+/;

    /* Differentiate between IE (detection via ActiveXObject)
     * and the rest (detection via navigator.plugins) */
    if (window.ActiveXObject) {
        var plugin_list = {
            flash: 'ShockwaveFlash.ShockwaveFlash.1',
            pdf: 'AcroPDF.PDF',
            silverlight: 'AgControl.AgControl',
            quicktime: 'QuickTime.QuickTime'
        }

        for (var plugin in plugin_list){
            var version = msieDetect(plugin_list[plugin]);
            if (version){
                var version_reg_val = version_reg.exec(version);
                found[plugin] = (version_reg_val && version_reg_val[0]) || ''
                    ;
            }
        }

        if (navigator.javaEnabled()){
            found['java'] = '';
        }
    } else {
        var plugins = navigator.plugins;
        var reg = /Flash|PDF|Java|Silverlight|QuickTime/;
        for (var i = 0; i < plugins.length; i++) {
            var reg_val = reg.exec(plugins[i].description);
            if (reg_val){
                var plugin = reg_val[0].toLowerCase();
                /* Search in version property, if not available concat name
                 *  and description and search for a version number in there
                    */
                var version = plugins[i].version ||
                    (plugins[i].name + ' ' + plugins[i].description);
                var version_reg_val = version_reg.exec(version);
                if (!found[plugin]) {
                    found[plugin] = (version_reg_val && version_reg_val[0])
                        || '';
                }
            }
        }
    }

    return found;

    /* Return version number if plugin installed
     * Return true if plugin is installed but no version number found
     * Return false if plugin not found */
    function msieDetect(name){
        try {
            var active_x_obj = new ActiveXObject(name);
            try {
                return active_x_obj.GetVariable('$version');
            } catch(e) {
                try {
                    return active_x_obj.GetVersions();
```

```
58                  } catch (e) {
59                      try {
60                          var version;
61                          for (var i = 1; i < 9; i++) {
62                              if (active_x_obj.isVersionSupported(i + '.0')){
63                                  version = i;
64                              }
65                          }
66                          return version || true;
67                      } catch (e) {
68                          return true;
69                      }
70                  }
71              }
72          } catch(e){
73              return false;
74          }
75      }
76  }
```

A poucos anos, dois trabalhos utilizaram JavaScript de forma diferente para obter informações sobre o navegador e os usuários. No artigo de Mowery et al.[29], os autores implementaram e avaliaram a identificação do navegador através da análise do desempenho e do tempo de execução de JavaScript. Para tanto, utilizaram uma combinação de 39 diferentes *benchmarks* JavaScript, bem conhecidos e bem estabelecidos, e geraram um *fingerprinting* 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.

Já o trabalho de Mulazzani et al. [14] propõe uma pesquisa voltada a identificação da segurança de um navegador baseado no uso de um mecanismo de *fingerprinting* do JavaScript. Para tanto, os autores utilizaram o Test262[2], um conjunto de testes oficiais para ECMAScripts. Foram usados 11.148 casos de teste únicos para navegadores de desktop e 11.570 casos de teste para navegadores móveis. Os autores concluíram que um único caso de teste pode ser suficiente para distinguir dois navegadores específicos. Para tanto, basta um dos navegadores falhar em um caso particular de teste e o outro não. No exemplo apresentado no artigo, o Opera versão 11.64 só falhou em 4 dos mais de 10 mil casos testados, enquanto o Internet Explorer 9 falhou em quase 400 casos de testes.

3.4 Adobe Flash

O Adobe Flash é utilizado para entrega de um rico conteúdo Web, no intuito de atrair e envolver os usuários. Através dele é possível exibir animações e interfaces de aplicativos, que são implantadas imediatamente em todos os navegadores e

[2]http://test262.ecmascript.org

plataformas. Dentre as principais características relacionadas ao plugin Adobe Flash, pode-se destacar: (i) a entrega de um console de jogos com qualidade para o navegador, (ii) a produção de impressionantes experiências de mídia e (iii) a implantação de conteúdo dinâmico em um tempo de execução mais seguro.

Apesar dessas características atrativas aos usuários, Adobe Flash é uma tecnologia usada para *Website Fingerprinting*. A Listagem 3.4 ilustra um código Flash (ActionScript) para determinar o *Timezone* do computador.

Listagem 3.4: ActionScript para determinar o Timezone

```
1  public class TimeZoneUtil
2      {
3          import com.adobe.utils.DateUtil;
4
5          /**  List of timezone abbreviations and matching GMT times. **/
6          private static var timeZoneAbbreviations:Array = [
7              ...
8              /* Atlantic Standard/Daylight Time */
9              {abbr:"AST", zone:"GMT-0400"},
10             {abbr:"ADT", zone:"GMT-0300"},
11             ...      ];
12
13         /** Return local system timezone abbreviation.**/
14         public static function getTimeZone():String
15         {
16             var nowDate:Date = new Date();
17             var DST:Boolean = isObservingDTS();
18             var GMT:String = buildTimeZoneDesignation(nowDate, DST);
19
20             return parseTimeZoneFromGMT(GMT);
21         }
22
23         /** Determines if local computer is observing daylight savings time
               for US and London. **/
24         public static function isObservingDTS():Boolean
25         {
26             var winter:Date = new Date(2011, 01, 01); // after daylight
                   savings time ends
27             var summer:Date = new Date(2011, 07, 01); // during daylight
                   savings time
28             var now:Date = new Date();
29
30             var winterOffset:Number = winter.getTimezoneOffset();
31             var summerOffset:Number = summer.getTimezoneOffset();
32             var nowOffset:Number = now.getTimezoneOffset();
33
34             if((nowOffset == summerOffset) && (nowOffset != winterOffset)) {
35                 return true;
36             } else {
37                 return false;
38             }
39         }
40
41         /** Goes through the timze zone abbreviations looking for matching
               GMT time. **/
42         private static function parseTimeZoneFromGMT(gmt:String):String
43         {
44             for each (var obj:Object in timeZoneAbbreviations) {
45                 if(obj.zone == gmt){
```

```
46                      return obj.abbr;
47                  }
48              }
49              return gmt;
50          }
51
52          /** Method to build GMT from date and timezone offset and accounting
                for daylight savings. **/
53          private static function buildTimeZoneDesignation( date:Date, dts:
                Boolean  ):String
54          {
55              if ( !date ) {   return "";   }
56
57              var timeZoneAsString:String = "GMT";
58              var timeZoneOffset:Number;
59
60              // timezoneoffset is the number that needs to be added to the
                    local time to get to GMT, so
61              // a positive number would actually be GMT -X hours
62              if ( date.getTimezoneOffset() / 60 > 0 && date.getTimezoneOffset
                    () / 60 < 10 ) {
63                  timeZoneOffset = (dts)? ( date.getTimezoneOffset() / 60 ):(
                        date.getTimezoneOffset() / 60 - 1 );
64                  timeZoneAsString += "-0" + timeZoneOffset.toString();
65              } else if ( date.getTimezoneOffset() < 0 && date.timezoneOffset /
                     60 > -10 ) {
66                  timeZoneOffset = (dts)? ( date.getTimezoneOffset() / 60 ):(
                        date.getTimezoneOffset() / 60 + 1 );
67                  timeZoneAsString += "+0" + ( -1 * timeZoneOffset ).toString()
                        ;
68              } else {
69                  timeZoneAsString += "+00";
70              }
71
72              // add zeros to match standard format
73              timeZoneAsString += "00";
74              return timeZoneAsString;
75          }
76      }
```

Fora informações “convencionais”, Adobe Flash pode ser usado em outras atividades. O estudo de Soltani et al. [30] observou um abuso de Cookies Flash, também chamados de objetos em locais compartilhados (LSO - *Local Shared Objects*), uma vez que Cookies HTTP previamente removidos foram regenerados. A técnica ficou conhecida como *respawning* (reaparecimento). No trabalho, 54 dos 100 sites mais populares (de acordo com a Quantcast) armazenavam Cookies Flash. Além disso, eles analisaram o *respawning* e descobriram que vários sites, incluindo aol.com, regeneravam Cookies HTTP previamente removidos através de Cookies Flash.

A Figura 3.3 ilustra as etapas do processo de *respawning* de Cookies Flash. Sempre que um usuário visita um site que usa Cookies, o site emite um ID e o armazena em vários mecanismos de armazenamento, incluindo Cookies, LSOs e armazenamento local. Na Figura 3.3a, o valor 123 é armazenado em Cookies HTTP e Flash. Quando o usuário remove os Cookies HTTP (Figura 3.3b), o site

reaparece com uma cópia do Cookie com o mesmo valor (123) através da leitura do valor (ID) em um Cookie Flash que o usuário pode não conseguir remover (Figura 3.3c).

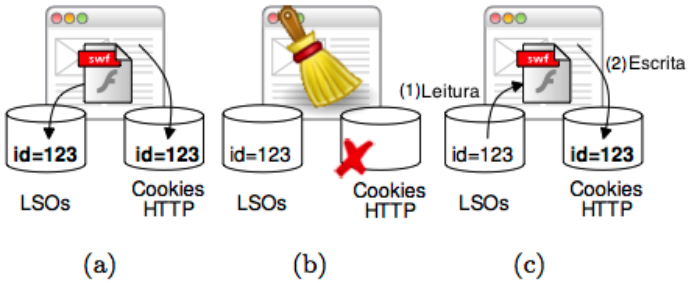

Figura 3.3: Exemplo de *Respawning* (a) a página Web armazena um Cookie HTTP e um Cookie Flash (LSO), (b) o usuário remove o Cookie HTTP, (c) a página Web "reaparece" com o Cookie HTTP copiando o valor do Cookie Flash. Fonte: [26]

Embora o uso de Cookies Flash tenha sofrido um declínio de uso e os sites que o empregam não sejam mais identificados de forma global, o *fingerprinting* baseado em Flash ainda é ameaça [31].

3.5 WebGL

WebGL é uma API multiplataforma que traz a linguagem OpenGL ES 2.0 para a Web, de forma a suportar desenhos 3D dentro do HTML [32]. Desenvolvida pelo Khronos Group[3], ela fornece uma API JavaScript para renderização de gráficos em 3D em um elemento Canvas da HTML5. Em outras palavras, oferece suporte para renderização de gráficos 2D e 3D. Atualmente, WebGL é suportado por todos os navagedores, mas somente está habilitado no Chrome, Firefox e Opera. O Safari vem com o WebGL desativado por questões de segurança, como explicado a seguir.

O site Browserleaks [33] disponibiliza um exemplo de JavaScript com WebGL que mostra como obter identificadores do navegador (Listing 3.5).

Listagem 3.5: Código exemplo de como identificar um navegador via WebGL [33]

```
1 var context = webgl_detect(1);
2 if (context)  {
3     var gl = context.gl;
4
5     // Getting WebGL Context Name:
```

[3]http://www.khronos.org

```
6       console.log("WebGL Context Name: " + context.name);
7
8       // Getting WebGL identifiers:
9       console.log("WebGL Version: " + gl.getParameter(gl.VERSION));
10      console.log("Shading Language Version: " + gl.getParameter(gl.
            SHADING_LANGUAGE_VERSION));
11      console.log("WebGL Vendor: " + gl.getParameter(gl.VENDOR));
12      console.log("WebGL Renderer: " + gl.getParameter(gl.RENDERER));
13
14     // Enumeration of supported WebGL Extensions:
15      var ext = [];
16      try {
17          ext = gl.getSupportedExtensions();
18      } catch(e) {}
19      var ext_len = ext.length;
20      if (ext_len) {
21          var ext_list = "";
22          for(var i=0;i<ext_len;i++) {
23              if (ext_list.length) {
24                  ext_list += "; ";
25              }
26              ext_list += ext[i];
27          }
28          console.log("Supported WebGL Extensions: " + ext_list);
29      }
30
31      // Some variables can be obtained only through the specific extensions
            ...
32      // Getting Max Anisotropy:
33      var max, e = gl.getExtension("EXT_texture_filter_anisotropic") || gl.
            getExtension("WEBKIT_EXT_texture_filter_anisotropic") || gl.
            getExtension("MOZ_EXT_texture_filter_anisotropic");
34      if (e) {
35          max = gl.getParameter(e.MAX_TEXTURE_MAX_ANISOTROPY_EXT);
36          if (max === 0) {
37              max = 2;
38          }
39      } else {
40          max = "Not available";
41      }
42      console.log("Max Anisotropy: " + max);
43  }
```

Em estudo de 2011, Forshaw [7] descobriu que existe uma série de problemas de segurança graves com a especificação e implementação do WebGL. Estas questões podem permitir que um invasor forneça um código malicioso, através de um navegador, que permita ataques contra os drivers de GPU e gráficos. Estes ataques podem inutilizar o processamento de toda a máquina. Além disso, os perigos trazidos pela WebGL incluem ataques de negação de serviço [34].

No quesito *Fingerprinting*, Forshaw [7] afirma que dentro do contexto de WebGL é possível se obter parâmetros de identidade do navegador, tais como: nome do fabricante, nome do navegador, motor de renderização e outras informações. Assim, os navegadores que permitem WebGL por padrão podem colocar seus usuários em risco. Uma contramedida ofertada pelos navegadores para mitigar os padrões de mau comportamento e incidentes graves com WebGL consiste

em apenas permitir acesso um conjunto de placas gráficas listadas em uma *whitelist*.

Num estudo posterior [8], foi descoberta uma vulnerabilidade que permite que qualquer imagem de vídeo que fosse exibida no sistema pudesse ser roubada por um atacante, lendo dados não inicializados da memória gráfica. Essa vulnerabilidade não se limita ao conteúdo WebGL, mas inclui outras páginas Web, o computador do usuário e outras aplicações. A Figura 3.4 ilustra essa vulnerabilidade e suas consequências.

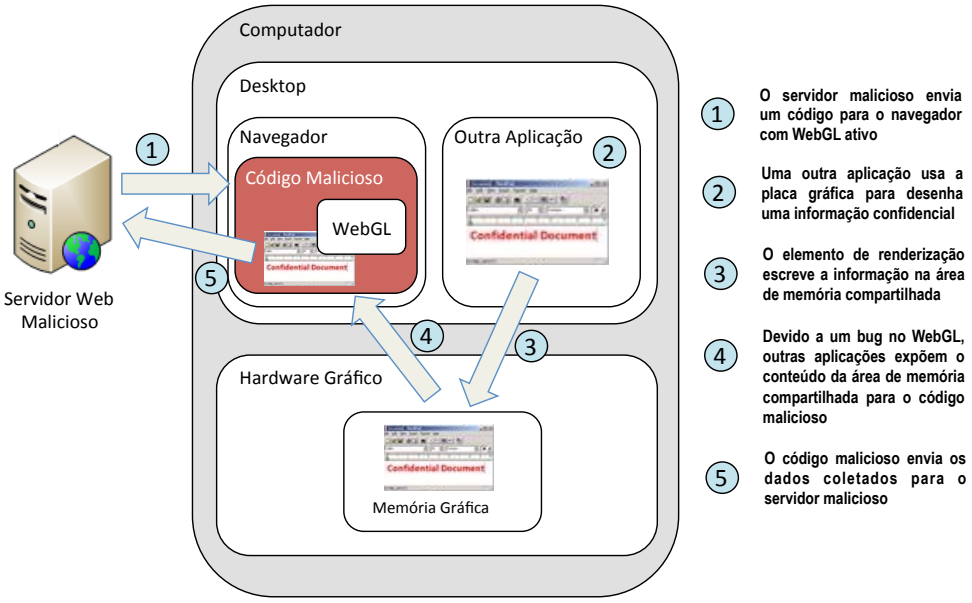

Figura 3.4: Exemplo de um ataque através do WebGL. Fonte: [8]

3.6 CSS

O *Cascading Style Sheets* (CSS) é uma linguagem de estilo utilizada para descrever a aparência e a formatação de um documento escrito em uma linguagem de marcação [35]. CSS foi projetada para permitir a separação do conteúdo do documento de apresentação de documentos, através de propriedades de estilo que incluem elementos de layout, cores, fontes, entre outros. Além disso, é independente da HTML e pode ser usado com qualquer linguagem de marcação baseada em XML. A separação do código HTML do CSS faz com que seja mais fácil de manter e melhorar a acessibilidade, proporcionando maior flexibilidade e controle na especificação de características de apresentação dinâmica. Em termos gerais, CSS é a parte do código que de fato da forma ao conteúdo.

No que tange *Fingerprinting*, CSS tem potencial para obter vários tipos de informação diferente dos tradicionais dados de fontes. A listagem 3.6 ilustra a técnica de Grossman [36] para obter o histórico do navegador via CSS (*browser history attack*), que permite vazar informações sensíveis sobre a privacidade e origem dos usuários. A técnica de Grossman verifica a cor de um link. Desta forma, o link visitado pelo estilo do usuário (*CSS user style*) é substituído por um estilo personalizado (como, por exemplo, cor rosa nos links). Um script é então usado para gerar dinamicamente links na página, potencialmente escondidos do usuário. Estes são comparados com o link rosa previamente substituídos. Se uma correspondência for encontrada, um invasor sabe que o site estava presente no histórico do navegador.

Listagem 3.6: Técnica de Grossman (*Browser History Attack* via CSS)

```
var agent = navigator.userAgent.toLowerCase();
var is_mozilla = (agent.indexOf("mozilla") != -1);

// popular websites. Lookup if user has visited any.
var websites = [ "http://ha.ckers.org", "http://mail.google.com/", ... ];

/* prevent multiple XSS loads */
if (! document.getElementById('xss_flag')) {

  var d = document.createElement('div');
  d.id = 'xss_flag';
  document.body.appendChild(d);

  var d = document.createElement('table');
  d.border = 0;
  d.cellpadding = 5;
  d.cellspacing = 10;
  d.width = '90%';
  d.align = 'center';
  d.id = 'data';
  document.body.appendChild(d);

  document.write('');
  for (var i = 0; i <> '');

  /* launch steal history */

  if (is_mozilla) {
    stealHistory();
  }
}

function stealHistory() {

  // loop through websites and check which ones have been visited
  for (var i = 0; i < websites.length; i++) {
          var link = document.createElement("a");
          link.id = "id" + i;
          link.href = websites[i];
          link.innerHTML = websites[i];
          document.body.appendChild(link);
          var color = document.defaultView.getComputedStyle(link,null).
              getPropertyValue("color");
```

```
43            document.body.removeChild(link);
44  // check for visited
45        if (color == "rgb(0, 0, 255)") {
46            document.write('' + websites[i] + '');
47         } // end visited check
48
49    } // end visited website loop
50  } // end stealHistory method
```

3.7 Modernizr

Modernizr[4] é uma pequena biblioteca JavaScript (7kb na versão comprimida) criada por Faruk Ates e Paul Irish. Esta biblioteca verifica quais funcionalidades o navegador suporta como, por exemplo, HTML5 e CSS3.

Embora não esteja ligada a artefatos conhecidos, ela prima pela compatibilidade da aplicação em diversos agentes Web, o que pode ser então empregado para *fingerprinting*. Por exemplo, o script abaixo é um trecho de código para verificar se o flash está ou não instalado no computador do usuário.

Listagem 3.7: Função para verificar se o Browser está com o plugin de flash instalado

```
1
2  Modernizr.on(feature,cb)
3
4  Modernizr.on('flash', function( result ) {
5    if (result) {
6     // the browser has flash
7    } else {
8      // the browser does not have flash
9    }
10 });
```

3.8 Silverlight

Silverlight é um framework criado pela Microsoft para o desenvolvimento e execução de aplicações ricas para a Internet, com recursos e propostas similares ao Adobe Flash. De acordo com a Microsoft [37], o Silverlight é um instrumento poderoso de desenvolvimento para a criação de experiências atrativas ao usuário, interativas para Web e aplicações móveis. Seu ambiente de execução está disponível por meio de um plugin gratuito para navegadores Internet Explorer, Mozilla Firefox e Google Chrome, que executam sob o sistema operacional Windows e Mac OS X, além de plataformas móveis como Windows Mobile e Symbian.

[4] https://modernizr.com/docs/

Em termos técnicos, o Microsoft Silverlight é uma aplicação *cross-browser*, *cross-platform* do framework .Net.

As primeiras versões do Silverlight focavam no *streaming* de mídia, mas as versões atuais suportam multimídia, gráficos e animação, e dão aos desenvolvedores suporte para idiomas e ferramentas de desenvolvimento. Assim como Flash, permite a criação de jogos 3D acelerados via hardware [23].

Embora o Silverlight possua várias características importantes e atraentes para os usuários, este plugin também está vulnerável as técnicas de *device fingerprinting*. Assim como o Adobe Flash, Silverlight também é atraente para a *fingerprinting*, sendo capaz de obter informações do sistema, como versão do sistema operacional, número de processadores, fuso horário, fontes instaladas, sistema, região, idioma do sistema operacional, entre outros. A listagem 3.8 apresenta uma função de Silverlight que lista os dispositivos disponíveis no computador. O código usa uma infraestrutura chamada WIA (*Windows Image Acquisition*), o que permite recolher (lista) dispositivos externos, tais como scanners, câmeras de vídeo e câmeras conectados ao computador local. Os resultados da função na listagem 3.8 são mostrados na Figura 3.5.

Listagem 3.8: Função ilverlight para lista os dispositivos de um computador

```
1  private void btnIterateWIA_Click(object sender, RoutedEventArgs e)
2  {
3      StringBuilder sb = new StringBuilder();
4      sb.AppendLine("List of available external devices and commands:");
5      using (dynamic DeviceManager = ComAutomationFactory.CreateObject("WIA.
           DeviceManager"))
6      {
7          var deviceInfos = DeviceManager.DeviceInfos;
8          for (int i = 1; i <= deviceInfos.Count; i++)
9          {
10             var IDevice = deviceInfos.Item(i).Connect();
11             var DeviceID = IDevice.DeviceID;
12             var DeviceName = IDevice.Properties("Name").Value;
13             var Commands = IDevice.Commands;
14             for (int j = 1; j <= Commands.Count; j++)
15             {
16                 var IDeviceCommand = Commands.Item(j);
17                 var CommandName = IDeviceCommand.Name;
18                 var CommandDescription = IDeviceCommand.Description;
19                 sb.AppendLine("    " + DeviceName + " (" + DeviceID + "), " +
                       CommandName + ": " + CommandDescription);
20                 // Execute with: IDevice.ExecuteCommand(IDeviceCommand.
                       CommandID);
21             }
22         }
23     }
24     MessageBox.Show(sb.ToString());
25 }
```

Figura 3.5: Exemplo de lista de dispositivos externos detectados

3.9 Discussão

Embora as tecnologias Web visem dinamizar o uso dos conteúdos em páginas na Internet, este Capítulo demonstrou que elas podem ser usadas para rastrear os usuários na Web.

Um bom exemplo é o HTML5, cujo objetivo é facilitar o acesso a variados recursos como, por exemplo, a disponibilidade de vídeos e imagens, mas que pode ser facilmente empregado na criação de scripts para *Website Fingerprinting*. Já tecnologias consagradas como JavaScript, notoriamente inseguras, tem o poder de "armar" scripts com a capacidade de acesso à propriedades e métodos de várias classes de objetos do navegador, como *navigator* e *screen*, e assim disponibilizarem sobre o dispositivo e seus usuários.

No entanto, tecnologias pouco conhecidas, ou que a primeira vista não pareçam ser problemáticas, podem e são úteis em *Website Fingerprinting*. Artefatos em Canvas permitem que empresas de publicidade identifiquem usuários na Web através de simples desenhos e linhas na tela do dispositivo. O mesmo pode ser dito, claro que com maior de grau de complexidade, sobre WebGL. CSS, Adobe Flash e Silverligth também possuem meios para fornecer informações sobre as configurações do usuário. Contudo, tais informações podem apresentar um nível elevado de periculosidade no que diz respeito a privacidade e segurança dos usuários.

Capítulo 4

Trabalhos Relacionados

Este Capítulo apresenta alguns trabalhos relacionados sobre o tema, em três seções, de forma a melhorar o entendimento em relação a *Website fingerprinting*. A primeira discute trabalhos que comprovam a existência de *Website fingerprinting* para identificação do usuário e/ou dispositivo bem como seu uso em soluções de segurança. A segunda seção apresenta trabalhos que desenvolvem soluções (contramedidas) que evitam ou inibem o uso dessas técnicas. A terceira, apresenta outras soluções que foram desenvolvidas tanto para identificação de *Website fingerprinting* como também para sua prevenção.

4.1 Identificando unicamente usuários e/ou dispositivos

Peter Eckersley [38], em seu artigo "How Unique Is Your Web Browser", relata sua experiência ao verificar o quanto a configuração de um navegador é única e facilmente empregada como mecanismo de rastreamento de usuários na Internet. A pesquisa investigou o grau em que os navegadores modernos estão sujeitos a *Website fingerprinting* através da análise de versões e informações de configuração que podem ser coletadas com ou sem autorização prévia dos usuários. Assim, Eckersley foi capaz de combinar todas as informações obtidas e gerar um identificador único para o dispositivo.

Para realizar essa pesquisa, foi desenvolvida uma página Web de teste **Panopticlick**[1], onde um algoritmo de *fingerprinting* é aplicado ao navegador do usuário e, consequentemente, um identificador único é gerado. Inicialmente, a pesquisa contou com 470.161 acessos de navegadores, cujos os usuários (participantes) foram informados do propósito ao visitarem a página.

[1] https://panopticlick.eff.org

Como resultado, foi possível gerar 83,6% de identificações únicas e apenas 5,3% de navegadores/usuários conseguiram se manter no anonimato. Também se observou que 94,2% dos navegadores que possuíam Adobe Flash ou tinham o suporte a Java habilitado foram identificados unicamente. Apenas 1,0% dos navegadores com Flash ou Java mantiveram o anonimato.

O autor é enfático ao afirmar que esses percentuais obtidos não são definitivos, pois quando acontecem mudanças no navegador (atualizações de versão ou plugins, desabilitação dos cookies, instalação de fontes, por exemplo) a identificação única gerada não era mais válida. Contudo, o preocupante é o fato de um algoritmo simples ser capaz de identificar todas as mudanças ocorridas, chegando a um percentual de 99,1% de acerto, com uma taxa de falso positivo de apenas 0,87%.

A pesquisa de Acar et al. [17], descrita no artigo "FPDetective: Dusting the Web for Fingerprinters", relatam o desenvolvimento de um framework, denominado FPDetective, para detecção de códigos de *fingerprinting*. Os componentes do framework são ilustrados na Figura 4.1 e seus componentes explicados a seguir.

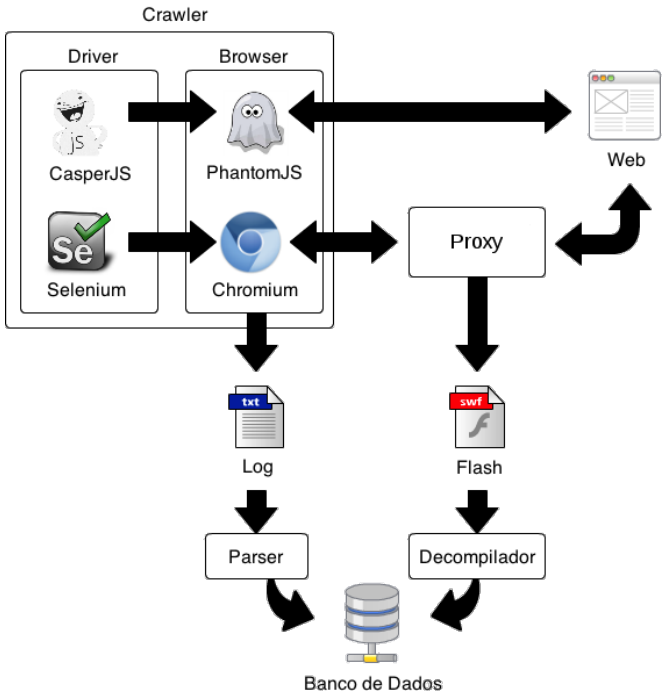

Figura 4.1: Arquitetura do Framework FPDetective

O **Crawler** é formado por dois navegadores, PhantomJS[2] e Chromium[3], instrumentados com CasperJS[4] e Selenium[5], respectivamente, para direcionar a coleta de páginas. O PhantomJS coleta códigos baseados em JavaScript enquanto o Chromium os baseados em Flash. O **Parser** é usado para extrair dados dos scripts coletados pelo Crawler e armazená-los em um banco de dados. A análise feita sobre arquivos Flash é estática e, por isso, todo o tráfego do Chromium é direcionado para um **Proxy** (formado pelo software mitmproxy[6], que é capaz de interceptar SSL, e pela biblioteca libmproxy, para analisar e extrair arquivos Flash). O **Decompilador**, que utiliza a biblioteca JPEXS Free Flash Decompiler[7], procura por chamada de funções (*enumerateFonts* e *getFontList*, por exemplo) de *fingerprinting*.

Ao realizar uma análise em grande escala, com milhões de sites Web populares da Internet, os autores descobriam que a adoção de técnicas de *Website fingerprinting* é muito maior do que os estudos anteriores haviam estimado, bem como a lista de códigos de *fingerprinting* conhecidos. Após a verificação realizada pelo FPDetective, foram encontrados 404 sites, que estão no Top 1 Milhão do alexa.com[8], que se utilizam de JavaScript para *fingerprinting*. Já em relação ao Flash foram encontrados elementos de *fingerprinting* em 145 sites no Top 100 mil da alexa.com, o que indica que *fingerprinting* baseado em Flash é mais prevalente. Os autores atribuem esse maior número de detecções em Flash devido às capacidades ampliadas para enumeração de fonte, detecção de proxy e compatibilidade com navegadores.

Mowery e Shacham [24] propuseram um novo sistema de identificação *fingerprinting*. Neste trabalho, denominado "Pixel Perfect: Fingerprinting Canvas in HTML5", os autores observaram que é possível relacionar o navegador às funcionalidades do hardware e do sistema operacional. Para tanto, desenvolveram um sistema *fingerprinting* onde quando um usuário visita uma página Web, o navegador é instruído a desenhar uma linha oculta de texto ou de gráfico 3D, que é então convertida em um sinal digital. Desta maneira, as diferenças na GPU ou no driver gráfico causam variações no token digital (ID gerado). O token pode ser armazenado e compartilhado com empresas de publicidade para identificar os usuários quando eles visitam sites Web afiliados. Um perfil pode ser criado como atividade de navegação de um usuário, permitindo que anunciantes direcionem

[2]http://phantomjs.org
[3]http://chromium.org
[4]http://casperjs.org/
[5]http://docs.seleniumhq.org/
[6]http://mitmproxy.org
[7]http://code.google.com/p/chromium/issues/detail?id=55084
[8]http://alexa.com

sua publicidade de acordo com as preferências do usuário.

Nessa mesma linha, os autores em Mowery *et. al.* [29] implementaram e avaliaram a identificação de um navegador por meio do desempenho e tempo de execução do motor JavaScript. No artigo, os autores criaram testes baseados nos benchmarks SunSpider[9] e V8[10], para avaliar o motor JavaScript, através de rótulos para gerar vetores de *fingerprinting* (características) do navegador. Assim, tais testes, chamados de primeiro nível, são aplicados na classificação da família/fabricante do navegador.

Como resultado, o conjunto de dados fornecidos pelos benchmarks foi suficiente para criar n-vetores *fingerprinting* para as principais versões do Chrome, Internet Explorer, Safari e Opera. Na prática, para cada navegador foram usados 39 diferentes benchmarks JavaScript, onde o vetor resultante é a *fingerprinting* de primeiro nível a partir dos padrões de tempo de execução. Dos 1015 acessos, 810 foram classificados corretamente, permitindo uma acurácia de 79.8% na identificação do navegador. Os autores relatam que os métodos são robustos o suficiente para detectar, de forma confiável, o perfil de execução do JavaScript para cada uma dessas 23 versões de navegadores. Além disso, pode funcionar para distinguir entre os navegadores e suas revisões, bem como identificar o sistema operacional e a arquitetura do dispositivo utilizado.

No que diz respeito aos problemas, os autores relatam erros de diagnóstico nas versões menores dos navegadores Firefox (versões 3.6 e 4.0). Tais problemas foram solucionados criando uma *fingerprinting* de segundo nível, constituída por 37 testes, mas somente aplicada se no teste de primeiro nível foi detectado um navegador da família do Firefox. Os autores não descrevem valores referentes a precisão na identificação da arquitetura do processador.

Boda *et. al* [39], no artigo "User Tracking on the Web via Cross-Browser Fingerprinting", propõem e avaliam um novo método de *fingerprinting*, independente do navegador, capaz de gerar um identificador único usando algoritmos JavaScript do lado do servidor. Assim, similarmente ao Panopticlick, os autores desenvolveram um site Web para teste, chamado Portal PET Internacional[11]. Além disso, como medida de comparação, os autores discutem melhorias no algoritmo utilizado na pesquisa do Panopticlick.

A ideia do trabalho foi mostrar que parte do endereço IP, a disponibilidade de um conjunto específico de fontes, o fuso horário e a resolução da tela são suficientes para identificar a maioria dos usuários que utiliza os cinco navegadores

[9] https://www.webkit.org/perf/sunspider/sunspider.html
[10] http://v8.googlecode.com/svn/data/benchmarks/v7/run.html
[11] http://pet-portal.eu/fingerprint/

mais populares atualmente. Além disso, que as cadeias de *user-agent* são bastante eficazes, mas são identificadores frágeis de uma instância do navegador.

Assim, um novo algoritmo *fingerprinting*, baseado unicamente em JavaScript, foi elaborado para incorporar a detecção de fontes como sua técnica principal. Os experimentos preliminares mostraram resultados promissores e para comprovar a viabilidade de tal técnica foi construído uma base de dados fingerprinting para uma análise posterior.

Como resultado, os autores perceberam que a correlação de listas de fontes e *UserAgents* proporcionaram uma base sólida para a identificação única dos sistemas operacionais Windows e Mac. De forma semelhante ao algoritmo de Panopticlick, o método criado pode controlar eficientemente mudanças no endereço IP dinâmico (por exemplo, quando um usuário se reconecta ou navega entre redes) e distinguir entre diferentes PCs atrás de um NAT (*Network Address Translator*). No entanto, em contraste com o método do Panopticlick, a *fingerprinting* elaborada é resistente a atualizações do computador e do navegador, troca de navegadores, (des) instalação de plugins e exclusão do *user-agent*.

Yen *et. al.* [40] relatam, em seu artigo "Host Fingerprinting and Tracking on the Web: Privacy and Security Implications", um estudo em grande escala para quantificar a quantidade de informação que pode ser revelada por um servidor, podendo implicar na privacidade e segurança das informações do usuário. A pesquisa analisa um conjunto de dados anônimos (Endereços IP, Cookies, Ids de login do usuário) coletados utilizando serviço de Web-mail Hotmail e do motor de busca Bing, com o intuito de demonstrar as implicações de segurança e privacidade de *host-tracking* nesses cenários.

Os autores verificaram que mesmo um usuário que realiza a limpeza de seus Cookies ou que utiliza a navegação privada pode ser rastreado. Descobriram também que ataques maliciosos podem estar sendo camuflados por mais de 75.000 contas *bot* que encaminham Cookies para locais distribuídos, através da análise de *One-time Cookies* (Cookies que parecem ser anônimos). Também examinam os padrões de mobilidade, por meio das faixas de IP, e assim estabeleceram um perfil de mobilidade do usuário capaz de detectar atividades incomuns e ataques.

Os resultados finais mostram que entre 60% a 70% dos valores do *user-agent* podem identificar com precisão visitantes. O uso de *user-agent* combinado com endereços IP tem uma entropia melhor na classificação de usuários do que a combinação entre plugins, resolução de tela, fontes e fuso horário. Além disso, o uso de *One-time Cookies* permite um reconhecimento de 88,00% dos usuários que voltam a um serviço. Vale ressaltar que entre estes usuários, 33% fizeram um esforço para preservar a sua privacidade como limpar os Cookies ou utilizar o modo de navegação privada.

Jang *et. al.* [41], relatam no artigo "An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Application", um estudo empírico da prevalência de fluxos de informações relacionados a roubo de Cookies, sequestro de sessão, *sniffing* de histórico e comportamentos de rastreamento, que tendem a violar a privacidade do usuário. Para tanto, implementaram uma nova versão do motor JavaScript dentro do navegador Chrome. O estudo utilizou 50.000 sites do alexa.com.

A pesquisa mostrou que o serviço de sites Web populares, incluindo o TOP 100 do alexa.com, apresentam fluxos que violam a privacidade para filtrar informações sobre os usuários como, por exemplo, o comportamento de navegação. Os autores encontraram instâncias de geolocalização, além de identificar várias agências de publicidade de terceiros utilizando as informações de Cookies. Também descobriram que: (i) vários sites Web fazem uso de informações do histórico como mecanismo *fingerprinting*; (ii) sites populares, como o da Microsoft e $huffingtonpost.com$, tem infraestrutura para rastrear os movimentos e cliques dos mouses dos usuários. Finalmente, descobriram que muitos sites que manipulam os fluxos de informação para violar a privacidade construíram sua própria infraestrutura, e não usam soluções como ClickTale[12], Tynt[13], Tealium[14] ou Beencounter[15].

Assim, o estudo mostra que as aplicações Web 2.0 populares, os chamados mashups[16], possuem agregadores e segmentadores de anúncios, além de estarem repletos de diferentes tipos de fluxos para violar a privacidade dos usuários.

Olejnik *et. al.* [42], no artigo "On the uniqueness of Web browsing history patterns", relatam uma pesquisa realizada em grande escala, com cerca de 368.284 usuários, para detectar os históricos de navegação dos usuários como ferramenta *fingerprinting*. Para o experimento utilizou-se o CSS e coleta dos sites alexa.com e quantcast.

Os resultados mostram que para a maioria dos usuários (69%), o histórico de navegação é único e que os usuários para os quais poderíamos detectar pelo menos quatro sites visitados foram identificados exclusivamente por suas histórias em 97 por cento dos casos. Os autores observaram uma taxa significativa de estabilidade nas *fingerprinting* no histórico do navegador. Ao repetir os testes, 38% das *fingerprinting* são idênticas ao longo do tempo e os diferentes foram

[12]http://www.clicktale.com
[13]http://www.tynt.com
[14]http://tealium.com
[15]http://www.beencounter.com/
[16]Mashups são diferentes serviços que podem funcionar simultaneamente com outros como, por exemplo, Google Maps

correlacionadas com conteúdo dos históricos originais, indicando estáticas preferências de navegação. O teste de 50 páginas Web foi o suficiente para ter uma acurácia de *fingerprinting* de 42% dos usuários no banco de dados, aumentando para 70%, com 500 páginas Web.

Yu *et. al* [43], no artigo "Identifying Webbrowsers in Encrypted Communications", tratam outra forma de identificar unicamente um usuário utilizando informações do tráfego de rede. Em linhas gerais, o artigo apresenta que as características de *fingerprinting* de um navegador também podem ser recolhidas apenas verificando os dados do tráfego de rede gerado ao se navegar em um determinado site Web.

Foram coletados dados do tráfego de rede gerados ao navegar na página inicial dos sites mais populares. Com base nestes dados, os autores mostram que as características de *fingerprinting* podem ser inferidas com alta precisão. Entre essas características, o tipo de navegador pode ser identificado com mais de 70% taxa de precisão. O estado de utilização de plugins populares como JavaScript e flash também pode ser identificado com precisão.

A pesquisa mostra que os dados de tráfego são seriamente afetados por características específicas do navegador. Isto sugere que ao realizar *fingerprinting* em um site Web com base em dados de tráfego, o navegador desempenha um papel muito importante e precisa ser considerado em trabalhos futuros.

4.2 Contramedidas

Seguindo a máxima de que a melhor solução para um problema geralmente é a mais simples, a melhor contramedida com *Website fingerprinting* é o cabeçalho *Do-Not-Track* (DNT), uma proposta de campo adicional no cabeçalho HTTP, através do qual é possível que uma aplicação Web ou um usuário solicite a desativação do seu rastreamento.

Definido inicialmente como um Draft da IETF (*Internet Engineering Task Force*) [44], DNT está sendo padronizado pelo W3C com o nome "Tracking Preference Expression" [45]. Em linhas gerais, é um mecanismo de escolha do consumidor que abrange todas as formas de monitoramento de terceiros, seja para publicidade, análise ou qualquer outra finalidade. O primeiro navegador a implementar o recurso foi o Mozilla Firefox, mas hoje todos os fabricantes oferecem suporte ao DNT.

Sob um prisma mais técnico, a Eletronic Frontier Foundation [46] define DNT como um mecanismo para proteger a privacidade on-line, uma vez que os anunciantes se utilizam de tecnologias de rastreamento cada vez mais sofisticadas. Assim, uma vez que o usuário configura seu navegador, todas as suas solicitações

enviaram os cabeçalhos DNT indicando que aquele computador (usuário) não deseja ser rastreado. Atualmente, várias empresas implementam a política Do Not Track em seus sites. BlueCava, DailyMail, Pinterest e Twitter são alguns exemplos [47]. Entretanto, por ainda não ser um padrão definitivo, muito menos acordado oficialmente, dificilmente DNT se tornará uma solução efetiva.

Deste modo, esta seção relatará o que está sendo discutido e utilizado como contramedida para o rastreamento de informações tanto no âmbito acadêmico quanto comercial.

4.2.1 Rede e Navegador TOR

Tor (*The Onion Router*)[17] [48, 49] é uma rede de túneis virtuais que permite aos usuários e grupos se comunicarem de forma segura, aumentando assim a segurança e a privacidade na Internet.

TOR redireciona o tráfego Internet por meio de uma rede mundial de voluntários, formada por mais de cinco mil nós livres, com a finalidade de ocultar a localização do usuário e realização de vigilância de rede ou de análise de tráfego. Com isso, torna mais difícil rastrear o tráfego de determinado usuário na Internet, ou seja, visitas a sítios Web, emails, mensagens instantâneas e outras formas de comunicação são protegidas, bem como a liberdade e habilidade para conduzir comunicação confidencial.

Um *draft* do projeto TOR [50] enumera as defesas que o navegador TOR oferece contra as técnicas de *fingerprinting*. São elas: (i) não habilitação de plugins; (ii) não permite o uso de WebGL; (iii) não permite a identificação das fontes do navegador; (iv) impede a busca de informações como resolução do desktop, consultas CSS e sistema de cores; (v) controle dos cabeçalhos HTTP e *user-agent*; (vi) não avaliação do desempenho do motor JavaScript; (vii) desativação de diversas preferências do HTML5.

Recentemente, o trabalho de Wang e Goldberg [49] - "Improved Website Fingerprinting on Tor" - demonstra que existem técnicas *fingerprinting* que podem ser mais potentes contra TOR do que se pensava anteriormente. Apesar do TOR oferecer um acesso seguro a seus usuários, a pesquisa mostra que um atacante pode tentar comprometer a privacidade de um cliente navegando na Web por meio da observação de padrões em sua sequência de pacotes. Um site Web pode vir a ser identificado com exclusividade a partir da ordem, direção e tamanho dos pacotes utilizados para carrega-lo, permitindo que um observador no lado do cliente possa identificar o servidor de destino de um cliente TOR, violando assim a privacidade se poderia esperar.

[17] O termo "onion routing" refere-se as camadas de aplicação de criptografia, comparadas com as camadas de uma cebola, usadas para tornar anônima a comunicação.

Os resultados experimentais com acessos reais demonstraram que um invasor pode potencialmente monitorar ou bloquear acessos a um local específico em TOR com altíssima precisão. Para demostrar isso, foram criados novos conjuntos de métricas para descrever a semelhança entre duas instâncias de tráfego. Elas são derivadas a partir de observações sobre a forma como um site é carregado. Como resultado, usando essas métricas, foram atingidas taxas de 95% de precisão e 0,06% de falsos positivos para vários sites.

Os autores utilizaram uma função que, dada a dois casos de tráfego, produz um valor que descreve como semelhante essas duas instâncias são. Esta medida de semelhança pode ser utilizada em SVM (*Support Vector Machine*) para classificar ocorrências de trânsito em as classes correspondentes ao local a partir do qual eles foram coletados.

4.2.2 Extensões e plugins

Esta subseção descreve alguns exemplos de extensões e plugins contra *fingerprinting*. Vale lembrar que existem inúmeras soluções disponíveis para os navegadores.

FireGloves é um plugin para Mozilla Firefox, cuja finalidade principal é impedir o rastreamento baseado em *fingerprinting* [51]. A fim de confundir os scripts de *fingerprinting*, o FireGloves, quando consultado, retorna valores aleatórios para determinados atributos, como: a resolução da tela, a plataforma na qual o navegador está em execução, o fornecedor do navegador e a versão. Adicionalmente, ele limita o número de fontes que uma aba do navegador pode carregar através de relatórios com valores falsos das propriedades dos elementos *offsetWidth* e *offsetHeight* da HTML, para evitar a detecção de fontes baseada em JavaScript.

O Ghostery é uma extensão voltada para privacidade na Internet que permite aos usuários detectarem e bloquearem objetos invisíveis e incorporados em páginas Web que recolhem dados sobre os hábitos de navegação do usuário [52]. O fabricante afirma que o Ghostery mostra todas as empresas que estão monitorando o usuário quando visita um site. Ghostery permite saber mais sobre as empresas e os tipos de dados que elas coletam dos usuários, bem como bloquear tais coletas.

O funcionamento do Ghostery se dá através da verificação do código HTML em cada página Web que o usuário visita, onde a existência de tags ou "trackers" colocados por uma empresa que trabalha com esse sítio é avaliada. Ele pode monitorar todos os diferentes servidores Web que estão sendo chamados a partir de uma determinada página e armazená-los em uma biblioteca de coleta de dados.

Se ele encontrar uma correspondência, Ghostery acrescenta a empresa a uma lista.

O ABP (*Adblock Plus*) é um filtro de conteúdo e um bloqueador de anúncios indesejados na Web [53]. Compatível com maioria dos navegadores Firefox (incluindo a versão para dispositivos móveis), Google Chrome, Internet Explorer, Opera e Safari, o ABP possui também uma versão para dispositivos Android. Dentre as principais características do ABP, destacam-se: (i) Suporte para bloquear as imagens de fundo; (ii) Assinaturas de filtros com endereço fixo e atualizações automáticas; (iii) Capacidade de ocultar elementos HTML, permitindo uma maior variedade de imagens a ser bloqueada; (iv) Capacidade de ocultar os anúncios em uma base por site, em vez de todo o mundo. Além do bloqueio de anúncios, o ABP também faz o rastreamento de anúncios.

O StopFingerprinting é uma extensão, disponível nos navegadores Mozilla Firefox e Google Chrome, que acessa propriedades do navegador e do sistema operacional do usuário e, consequentemente, envia os dados coletados para um servidor seguro no INRIA (Institute for Research in Computer Science and Automation) [54]. Os dados coletados somente são utilizados para fins de pesquisa dentro INRIA e não são compartilhadas com ninguém fora da INRIA. Em outras palavras, é uma extensão para proteção e análise de *fingerprinting* e sua função principal é recolher *fingerprinting* de navegadores, a fim de analisá-los.

Dentre as informações coletadas via StopFingerprinting, ignorando as comumente capturadas, destacam-se: (i) *Mime-types*; (ii) Constantes Matemáticas; (iii) informações das câmeras; (iv) Informações dos microfones, incluindo codecs, ganho, nível de silêncio, nível de supressão de ruído e muito mais; (v) Se existe um acelerômetro ativado e (vi) Se existem teclados virtuais e físicos.

O Lightbeam é um *add-on* para Firefox, no qual, uma vez instalado e ativado, registra todos os Cookies de rastreamento salvos no computador do usuário [55]. Desta maneira, exibe um gráfico das interações e conexões de sites visitados e os locais de rastreamento para onde eles fornecem estas informações. De acordo com a Mozilla, através deste *add-on* os usuários podem ver onde e como eles estão sendo rastreados por anunciantes.

O funcionamento do Lightbeam é simples. Quando ativado e o usuário visita um site, o *add-on* cria uma visualização de todos os terceiros (empresas) que atuam nessa página em tempo real. A visualização padrão é chamada de gráfico de ponto de vista. Quando o usuário navega para um segundo site, o *add-on* destaca os terceiros que estão ativos lá e mostra aqueles que foram vistos em ambos os locais. A visualização cresce a cada site que o usuário visita e a todos os pedidos feitos a partir do navegador. Além da visão do gráfico, o usuário

também pode ver seus dados em uma visualização tipo relógio, para examinar as conexões ao longo de um período de 24 horas ou em uma lista detalhada de sites individuais.

4.3 Soluções Híbridas

Nikiforakis e Livshits [56], no artigo "PriVaricator: Deceiving Fingerprinters with Little White Lies", apresentam uma solução para dificultar o acesso as informações que os navegadores podem disponibilizar. Esta solução pode ser aplicada por meio de um proxy HTTP, uma extensão ou incorporada no próprio navegador.

Os autores partilham a visão de que o maior problema à privacidade usuário, via *fingerprinting*, não é a geração de uma identificação única, mas sim o fato de existir a capacidade de acessar/obter várias informações do usuário e sobre o usuário através de múltiplas vias e visitas. Neste sentido, o *Privaricator* tenta, por meio de randomizações parametrizadas, modificar as propriedades do navegador de forma que a cada visita a site Web com *fingerprinting* seja gerada uma resposta diferente e, consequentemente, um identificador diferente, frustrando assim as tentativas de rastreamento do usuário.

Como prova de conceito, os autores alteram o navegador Chromium, versão 34.0.1768.0, instrumentando o WebKit (motor de renderização) para incorporar várias funções de randomização. Sites como BlueCava[18], Coinbase[19] e PetPortal, bem como a biblioteca fingerprintjs[20], foram utilizados nos testes de validação.

Como resultado, os experimentos mostraram o quanto o *PriVaricator* foi capaz de enganar todos os *fingerprintings* da grande fração de combinações testadas. Para os sites BlueCava e Coinbase, os percentuais de usuários não identificados foram de 94,58% e 97,86%, respectivamente, mostrando o quanto as modificações realizadas são eficientes e quanto é frágil o sistema de identificação desses sites. Para a biblioteca fingerprintjs, 78,15% dos artefatos avaliados puderam ser ludibriados com a aleatoriedade, retornando diferentes conjuntos de plugins, o que em outras palavras significa diferentes identificações. Na contramão dos outros resultado, o PetPortal teve apenas 43,61% dos artefatos "enganados". Isso provavelmente se deve ao fato do PetPortal colocar mais peso na descoberta de fontes do que em plugins, além de ser executado no servidor e não no cliente.

Khademi [57] apresenta uma solução, denominada FPGuard, cuja finalidade é detectar e proteger os usuários contra o recolhimento de *fingerprinting* no momento da execução do conteúdo. Dada uma página Web em execução no nave-

[18] http://bluecava.com
[19] https://www.coinbase.com
[20] https://github.com/Valve/fingerprintjs

gador do usuário, o FPGuard monitora e registra sua atividade a partir de seu carregamento. Em seguida, extrai métricas relativas a cada método *fingerprinting* e constrói uma assinatura para a página Web. Após isso, usa vários algoritmos para distinguir páginas normais de *Website fingerprinting*.

Quando uma página é tida como *fingerprinting*, FPGuard notifica o usuário com um alerta e armazena a URL em uma lista negra. O usuário tem a opção de confiar na página e deixá-lo executar bem como impedi-la usando uma das técnicas de prevenção recomendadas. Para proteger os usuários contra *fingerprinting*, foram combinadas técnicas baseadas em randomização e filtragem.

A avaliação do FPGuard foi realiza usando o top 10K do alexa.com. Para automatizar o processo de visitar os sites e coleta de dados, os autores usaram *iMacros* (extensão para o Navegador Google Chrome) para a realização de tarefas repetidas tais como clicar em links e visitar uma série de sites automaticamente. Para coletar as métricas e analisá-las para cada sites, eles desenvolveram um script que, primeiramente, visita a página inicial e aguarda 5 segundos para o carregamento dos recursos. Após plena carga, FPGuard registra as métricas e identifica as atividades da página Web como *fingerprinting* ou normal.

Os autores conseguiram coletar métricas para 9264 sites. Como resultado, foram encontrados um pequeno número de sites que acessam todas as propriedades do navegador e objetos de tela através de JavaScript. Também verificaram que, embora muitos sites carreguem menos de 50 fontes (99,7% de 10K), alguns sites carregam mais de 300 fontes. Os autores descobriram que *fingerprinting* é explorado por sites de diferentes categorias, que vão desde relacionamentos a notícias.

O FPGuard descobriu que arquivos Flash suspeitos, que incluem métodos para enumerar as fontes do sistema, têm métodos para transferir as fontes coletadas para um servidor remoto ou um programa de JavaScript e armazenar as fontes como Cookies Flash. Finalmente, a sobrecarga de desempenho que o FPGuard impõe ao navegador é considerada desprezível.

O SHPF (*Session Hijacking Prevention Framework*), proposto por Unger *et. al.* [13], é um framework desenvolvido para combater o sequestro de sessões (*Session Hijacking*) em navegadores Web através de *fingerprinting*. SHPF protege especialmente contra sequestro de sessões bem como contra XSS (*Cross-site scripting*). A ideia do framework é: se o navegador do usuário muda de repente, por exemplo do Firefox no Windows 7 de 64 bits para um navegador Webkit baseado em Android em uma faixa de IP totalmente diferente, é de se supor que algum tipo de atividade maliciosa está acontecendo.

Assim, o framework SHPF consegue melhorar o gerenciamento de sessões HTTP(S) através de um *fingerprinting*. Para tanto, dois novos métodos baseados

em CSS e HTML5 foram desenvolvidos.

Em relação a implementação, o SHPF foi escrito em PHP5 e consiste de múltiplas classes que podem ser carregadas independentemente. A Figura 4.2 apresenta a arquitetura geral e as funcionalidades básicas do SHPF.

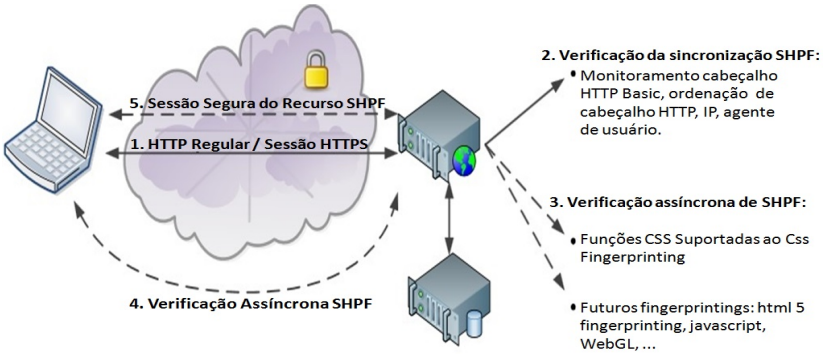

Figura 4.2: Arquitetura e funcionamento do SHPF. Fonte: [13]

As principais partes da estrutura do framework são chamadas de *features*. Uma *feature* é uma combinação de diferentes verificações para detecção e mitigação do sequestro de sessão. O protótipo implementa as seguintes *features*: monitoramento do cabeçalho HTTP, *fingerprinting* CSS e SecureSession (que implementa e amplia o protocolo SessionLock [13]. As *features* também são os meios de extender o framework e fornecer classes base para o desenvolvimento rápido de novas *features*. Na prática, uma *feature* consiste de um ou mais *checkers*, que são utilizados para executar certos testes. Existem dois tipos diferentes (ou classes) de *checkers*: os síncronos, que podem ser utilizados se os testes incluídos forem executados apenas a partir de dados existentes e são passivos por natureza, tais como solicitações HTTP; e os assíncronos, que são usados se os testes têm de solicitar ativamente alguns dado adicional do cliente e o framework tem que processar a resposta.

Como resultado, o SHPF é capaz de detectar e impedir ataques e tentativas de sequestro de vários tipos, tais como XSS e *sniffing* passivo na mesma rede Wi-Fi. Além disso, propõem dois novos métodos de *fingerprinting* de navegador baseados em HTML5 e CSS, cujo finalidade é detectar sequestro de sessão. SHPF pode ser configurado para ser executado com diferentes níveis, permitindo mais segurança para sessões sensíveis ou partes da sessão. Futuros métodos de *fingerprinting* podem ser incorporados facilmente em sua estrutura.

4.4 Discussão

A Tabela 4.1 apresenta, de forma resumida, uma comparação entre os trabalhos tanto de identificação quanto de mitigação de *Website fingerprinting*. Para tanto, ilustra: (i) o foco do trabalho (classificação); (ii) as tecnologias empregadas para realização do *fingerprinting*; (iii) as propriedades do navegador exploradas por cada um dos trabalhos; (iv) se um protótipo funcional foi disponibilizado; e (v) a fonte de dados usada na avaliação das soluções.

Tabela 4.1: Comparação entre Soluções

Artigo	Classificação	Tecnologias Empregadas	Propriedades do Navegador	Protótipo	Fonte de Dados
Panopticlick [38]	Identificação	JavaScript e Flash	Nav, Scr, TZ, SF, HTTP	Sim	*
FPDetective [17]	Identificação	JavaScript, Flash, HTML5 e CSS3	Nav, Scr, Win, SF	Sim	alexa.com
Pixel Perfect [24]	Identificação	HTML5 Canvas, WebGL, CSS3 e JavaScript	*	Sim	Mechcanical Turk
Fingerprinting in JavaScript [29]	Identificação	JavaScript	Nav	Não	Mechcanical Turk
SHPF [13]	Híbrido	HTML5, CSS e JavaScript	HTTP	Sim	*
Cross-Browser Fingerprinting [39]	Identificação	JavaScript	Nav, Scr, TZ, SF, HTTP	Sim	Panopticlick
Host Fingerprinting [40]	Identificação	*	HTTP	Não	Hotmail web-mail
Privacy-Violating [41]	Identificação	JavaScript	Não	Não	alexa.com
Web Browsing History [42]	Identificação	CSS	Hy	Não	Não
Identifying Webbrowsers [43]	Identificação	*	Não	Não	alexa.com
Privaricator [56]	Híbrido	JavaScript e HTML5 Canvas	Nav e Scr	Não	alexa.com, bluecava, coinbase, petportal e fingerprintjs
FPGuard [57]	Híbrido	JavaScript, Flash e HTML5 Canvas	Nav, Scr, SF, TZ, HTTP	Não	alexa.com
Determinando o Risco de Fingerprinting em Páginas Web	Alerta e Classificação de Risco	JavaScript	Win, Doc, Nav, Scr, Sil, Mod, WGL, Can	Não	Diversas

Propriedades do Navegador: Nav-objeto Navigator; Scr-objeto Screen; Win-objeto Window; TZ-timezone; Doc-objeto Document;
SF-sistema de fontes; HTTP-cabeçalho HTTP; HY-histórico;
Sil-Microsoft Silverligth; Mod-Biblioteca Modernizr; WGL-WebGL; Can-Canvas
*Não aplicável ou não informado

Vale ressaltar que os valores indicados nas quatro últimas colunas da Tabela 4.1 foram retirados dos trabalhos originais. Além disso, uma comparação precisa das soluções exigiria que todas fossem implementadas em um mesmo ambiente e submetidas a testes com o mesmo conjunto de sites Web.

Verificando a Tabela 4.1 em detalhes, é possível ratificar relações mais gerais. Primeiro, é interessante observar que existem poucas soluções acadêmicas focadas em combater *Website Fingerprinting*. Tirando os plugins e as duas soluções híbridas ([56] e [57]), todas as demais soluções focam provar a existência do fato.

As soluções que investigam a identificação do uso de *fingerprinting* criam, em geral, sua própria ferramenta (Website) enquanto as que mitigam focam na proteção individual dos usuários. Em outras palavras, nenhuma delas foca na proteção global dos usuários, como é o caso deste trabalho.

A abordagem proposta nesta dissertação investiga diretamente nos artefatos (scripts) os termos *fingerprinting* e após encontrá-los realizada a classificação pelo nível de severidade de risco (alta, média e baixa), baseando-se nos termos do dicionário previamente criado. Vale ressaltar que a solução proposta, diferente das demais, não se preocupa em identificar e/ou mitigar *fingerprinting*, mas sim assume que quase todas a páginas Web usam esse artifício (para variados fins) e, desta forma, visa classificar os riscos dos artefatos, alertando assim os usuários.

No critério tecnologias empregadas, é possível constatar o maciço uso de JavaScript como mecanismo de geração do *fingerprinting*. Mais especificamente, apenas três trabalhos não usam JavaScript. Além disso, os únicos três trabalhos que usam Flash, também usam JavaScript. Este trabalho segue a tendência e também faz uso de JavaScript.

Já sobre as propriedades, ficou claro que as soluções que empregam JavaScript e Flash acabam tendo uma maior penetração no navegador e, consequentemente, conseguem acessar mais propriedades do navegador e assim obter uma quantidade maior e mais detalhada de informações. O trabalho proposto emprega uma quantidade maior de termos (propriedades do navegador) para avaliação dos artefatos *fingerprinting*. A definição desses termos foi feita através de pesquisas na literatura que comprovam seu uso para *fingerprinting*. Tal fato, permitiu a criação de um dicionário de termos, os quais foram categorizados em três (3) níveis (alta, média e baixa). Além disso, o dicionário é flexível, ou seja, novos termos podem ser agregados sem nenhum empecilho.

Sobre a criação de protótipo, embora o foco da abordagem proposta aqui não tenha uma implementação formal, ela poderá ser agregada à uma aplicação, um plugin ou o módulo de um Proxy Web, permitindo verificar e classificar *fingerprinting* em páginas Web em tempo real. Além disso, percebe-se na Tabela que as soluções com protótipo implementado são as que fazem identificação de *fingerprinting*.

Por fim, no quesito fonte de dados, a solução proposta emprega cinco (5) bases de dados (Controle, Canvas, DMOZ, PhishTank e Alexa.com - na categoria brasil). A única solução com tamanha quantidade de fontes é o Privaricator.

Capítulo 5

Projeto e Implementação

Este Capítulo apresenta a visão geral da abordagem proposta para detecção de *Website Fingerprinting* a partir do conjunto de propriedades e métodos dos objetos HTML DOM capazes de discriminar padrões que compõem um artefato *fingerprint* e seu nível de risco à privacidade dos usuários.

5.1 Visão Geral

A Figura 5.1 apresenta a abordagem proposta, executada em quatro etapas: (1) coleta de páginas; (2) extração de scripts; (3) extração de termos e (4) classificação da severidade.

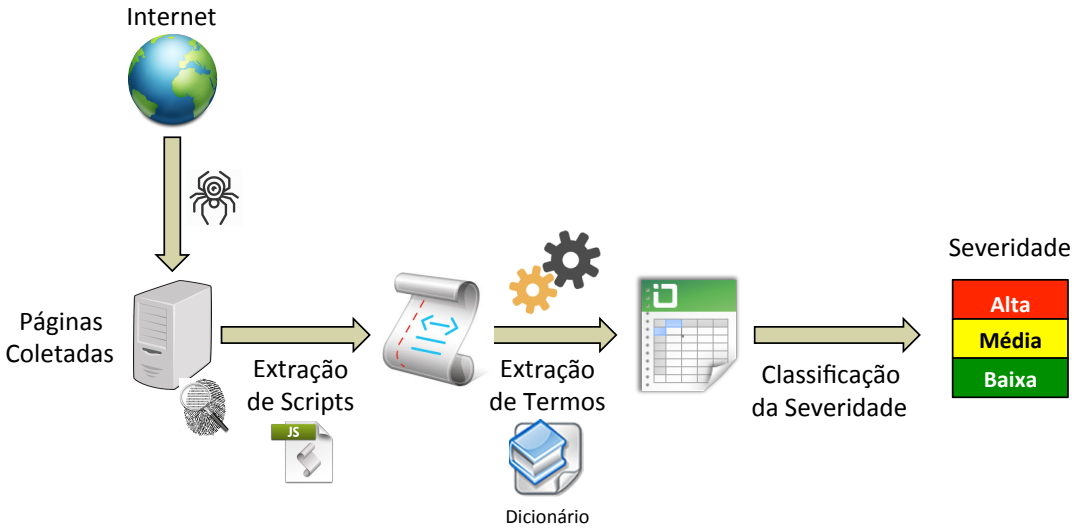

Figura 5.1: Visão geral da abordagem proposta

Em linhas gerais, a ideia é:

1. Coletar as páginas Web informadas (sementes) para avaliação;
2. Extrair os códigos em JavaScript de uma determinada página Web e sumarizalos em um único arquivo de script;
3. Extrair os termos (objetos, propriedades e métodos) relacionados aos artefatos conhecidos de *fingerprinting*. Para tanto, é utilizado um dicionário, previamente elaborado, contendo os principais objetos, propriedades e métodos usados em *Website Fingerprinting*;
4. Classificar a severidade ao qual o usuário está exposto ao acessar a página Web.

A intenção é manter o usuário informado sobre a existência de artefatos *fingerprinting* na página acessada e qual seu nível de severidade. Este alerta permitirá ao usuário ficar ciente da possibilidade de invasão da sua privacidade, incluindo a captura de dados e sua posterior utilização em atividades maliciosas.

5.1.1 Coleta das Páginas Web e Extração de Script

A primeira e segunda etapas da abordagem proposta correspondem a coleta das páginas Web e a extração dos scripts (JavaScripts) contidos nelas. Para executar essas funções, foi implementado um *crawler*, em linguagem PHP (versão 5.5.15), que para cada site visitado, coleta os scripts JavaScript existentes e os armazena em um único arquivo contendo todos scripts.

5.1.2 Extração de Termos

A terceira etapa da abordagem é a extração de termos, cuja finalidade é identificar os termos (objetos, propriedades e métodos) presentes no arquivo de **script** gerado na etapa anterior. Para esta etapa foi desenvolvido um script de extração na linguagem Python (versão 3.4) que realiza a leitura do arquivo único e extrai as propriedades e métodos de acordo com um dicionário previamente estabelecido.

A criação do dicionário de termos foi feita tomando-se como base pesquisa na literatura tanto de implementação (objetos, propriedades e métodos) quanto acadêmica (soluções e contramedidas). Em outras palavras, trabalhos, scripts e exemplos de códigos foram avaliados e os objetos, propriedades e métodos utilizados para realizar *fingerprinting* foram selecionados. A lista completa contendo 99 termos pode ser vista no apêndice A.

Em relação ao processo propriamente dito, a extração dos termos é feita através do uso de expressões regulares (REGEX), um excelente recurso para

casamento de padrões e amplamente utilizadas em conteúdo de texto, afirma Sudkamp [58]. O uso de REGEX permite que o script de extração seja capaz de buscar objetos, propriedades, atributos e métodos, possibilitando a descoberta de termos em até três níveis. A Tabela 5.1 apresenta o uso de expressões regulares na extração de termos.

Tabela 5.1: Expressões Regulares para Extração de Termos

Expressão Regular	Níveis
$("N2", r'([\w] * \.[\w]*)')$	Essa expressão regular busca por objetos que utilizam em sua sintaxe de chamada até dois níveis de acesso a propriedade. Exemplo: *navigator.userAgent*
$("N3", r'([\w] * \.[\w] * \.[\w]*)')$	Essa expressão regular busca por objetos que utilizam em sua sintaxe de chamada até três níveis de acesso a propriedade. Exemplo: *window.navigator.plugin*
$("N4", r'([\w] * \.[\w] * \.[\w] * \.[\w]*)')$	Essa expressão regular busca por objetos que utilizam em sua sintaxe de chamada até quatro níveis de acesso a propriedade. Exemplo: *window.navigator.plugin.name*

Vale ressaltar que os níveis de busca do REGEX estão de acordo com o conteúdo do dicionário de termos (vide apêndice A).

5.1.3 Classificação da Severidade

A quarta e última etapa é a classificação da severidade, cuja finalidade é definir o nível de risco (severidade) ao qual o usuário está exposto a acessar uma página Web. Embora tenha sido baseado na metodologia para estimação de riscos da OWASP, o modelo de classificação da severidade dos artefatos *fingerprinting* é mais simples e direto. A ideia adotada foi estimar o grau de envolvimento de elementos (objetos, propriedades, atributos e métodos) com ataques e intrusões na Web.

Para isso, foi realizada uma detalhada pesquisa na literatura acadêmica e na área de segurança da informação (site, fóruns de discussão, blogs de desenvolvedores, comunidades hackers, entre outros), onde cada objeto, propriedade, atributo e método listados no dicionário de termos foi coorrelacionado à ataques, vulnerabilidades e intrusões. Ou seja, como cada um desses elementos pode e é usado para explorar vulnerabilidades, violar privacidade, obter dados e realizar ataques.

Desta forma, amparados pela metodologia da OWASP, foram definidas três classes de severidade para os objetos, propriedades, atributos e métodos: Baixa,

Média e Alta.

Baixa

Na classificação Baixa encontram-se os elementos capazes apenas de fornecer informações sobre o dispositivo ou usadas somente para adaptações de conteúdo (tamanho da tela e a versão do navegador, por exemplo). Os objetos *Window*, *Document*, *Navigator*, *Screen* e quase todas as suas propriedades, atributos e métodos que se enquadram nessa classificação.

Vale ressaltar que nenhum trabalho acadêmico ou local pesquisado provou ou interligou qualquer um desses elementos com ataques ou invasões de privacidade. Por isso são considerados de baixo impacto como ameaça à privacidade. Contudo, no momento em que são combinados com outros elementos, podem compor uma poderosa arma de *fingerprinting* capaz de identificar unicamente o usuário e/ou dispositivo, conforme já demonstrado nos trabalhos de Eckersley [38], Acar [17], entre outros.

Média

Na classificação Média encontram-se os plugins *SilverLight* e *WebGL*, o uso da biblioteca *Modernizr* e a verificação de *plugins* e *mimeTypes*. Os plugins *SilverLight* e *WebGL* foram explicados e contextualizados no Capítulo 3. Já a biblioteca Modernizr é capaz de detectar automaticamente a disponibilidade de tecnologias presentes no navegador do usuário. Ela permite, por exemplo, detectar o suporte a geolocalização, ao elemento <canvas>, aos plugins WebGL e flash, bem como o armazenamento local e armazenamento de sessão. Unger *et. al* [13] afirma que o a biblioteca modernizr é uma poderosa tecnologia para obtenção de recursos disponíveis no navegador. Por fim, a verificação dos plugins e mimeTypes existentes no navegador pode fornecer informações relevantes a um atacante, como, por exemplo, dados dos plugins bancários ou alguma informação de um software específico. Khademi[57], Eckersley [38],[56] descreveram esses objetos como propensos a um alto/médio grau de ameaça.

Alta

A classificação Alta é composta apenas pelos elementos: *canvas.toDataUrl*(), *canvas.getImageData*(), *window.history*() e *MediaDevices.getUserMedia*(). O método *canvas.getImageData*() retorna um objeto *ImageData* que copia os dados de pixel para o retângulo especificado em área canvas. Na verdade, é preciso esclarecer que um objeto *ImageData* não é uma figura, e sim parte da área canvas e manipula a informação de cada pixel dentro daquele retângulo. Já o método *canvas.toDataURL*() permite obter o conteúdo da tela do navegador. Os dados retornados formam uma string que representa uma URL codificada que

contém os dados gráficos. Os autores Mowery[24], Nickforakis [56], Tillmann [59] e Acar [26] relatam em suas pesquisas o uso desses métodos para *fingerpriting*. Outra forma de demonstrar o uso dos métodos *getImageData*() e *toDataURL*() é a extensão para Chrome **CanvasFingerprintBlock**, que realiza o bloqueio dos métodos, enviando uma imagem em branco para o servidor solicitante.

O objeto *history* contém a lista das URLs visitadas pelo usuário dentro de uma janela do navegador. É parte do objeto *window*, possui a propriedade *length*, que retorna o número de URLs no histórico, e três métodos: *back*, que carrega a URL anterior no histórico; *forward*, que carrega a próxima URL no histórico; e *go*, que carrega uma URL específica do histórico. É importante notar que embora não seja um padrão definido, todos os navegadores suportam este objeto. Olejnik et al. [42] mostraram que o histórico no navegador por também ser usado como meio para realizar um *fingerprinting* sem a necessidade de ajuda do lado do cliente. Para tanto, eles analisaram uma base de dados gerada quando uma falha do CSS em relação ao histórico ainda estava presente em navegadores.

Por fim, o método *MediaDevices.getUserMedia*() solicita ao usuário permissão para usar um dispositivo de vídeo e/ou uma entrada de áudio, como uma câmera, compartilhar a tela e/ou um microfone. Tian et al. [60] apresentam múltiplos ataques usando *MediaDevices.getUserMedia*() que podem comprometer a confidencialidade e integridade dos usuários. Eles mostram como um atacante pode roubar dados de um usuário da tela para modificar suas contas de acesso a site populares.

5.1.4 Termos Investigados

A Tabela 5.2 apresenta 58 termos mais relevantes dos 99 identificados inicialmente definidos. A base de Controle foi usada para análise e delimitação dos termos. A descrição dos termos implementados para a classificação da severidade de *fingerprinting* nos sites Web pode ser vista no apêndice B.

Os termos foram separados por níveis (1, 2 e 3) de ameaça que podem causar ao usuário.

Tabela 5.2: Termos implementados

Níveis	Objetos
1	window.innerHeight; window.outerWidth; window.outerHeight; window.devicePixelRatio; document.domain; navigator.userAgent; navigator.appCodeName; navigator.appName; navigator.appVersion; navigator.appMinorVersion; navigator.buildID; navigator.browserLanguage; navigator.cpuClass; navigator.doNotTrack; navigator.language; navigator.onLine; navigator.oscpu; navigator.platform; navigator.userLanguage; navigator.product; navigator.productSub; navigator.securityPolicy; navigator.systemLanguage; navigator.vendor; navigator.vendorSub; navigator.geolocation; navigator.savePreferences; screen.availHeight; screen.availWidth; screen.availLeft; screen.availTop; screen.height; screen.width; screen.colorDepth; screen.pixelDepth; screen.bufferDepth; screen.deviceXDPI; screen.deviceYDPI; screen.logicalXDPI; screen.logicalYDPI; screen.systemXDPI; screen.systemYDPI; screen.updateInterval
2	document.referrer; document.cookie; Modernizr; Silverlight; WebGL; navigator.cookieEnabled; navigator.javaEnabled; mimeTypes; plugins; window.localStorage
3	navigator.getUserMedia; canvas.toDataURL; canvas.getImageData; window.history

Capítulo 6

Experimentos e Resultados

Este Capítulo descreve os passos executados para classificação dos websites quanto ao nível de severidade de *fingerprinting* em baixa, média ou alta. A primeira seção relata o protocolo experimental necessário (ambiente e base de dados), enquanto a segunda descreve o processo de aplicação da metodologia proposta e os resultados alcançados.

6.1 Protocolo Experimental

Para classificar os sites quanto ao nível de risco de suspeita *fingerprinting* é necessário a realização de vários experimentos com as base de dados coletadas empregando todos os 58 termos investigados.

6.1.1 Ambiente de Experimentação

Os experimentos realizados foram executados em duas máquinas. A primeira é uma estação de trabalho Intel Core i7, 3,4 Ghz, com 8 GB de memória RAM, 500 GB de disco, com sistema operacional Linux, distribuição Ubuntu 14.04. A segunda um notebook Core i5, com 8GB de memória RAM, 1Tera de HD com sistema operacional Windows 8.

6.1.2 Base de Dados

Para elaboração dessa dissertação foram utilizadas cinco (5) base de dados. A primeira, denominada **base de controle**, é formada por 250 websites classificados como *fingerprinting* nos artigos de Nickforakis [56], Khademi [57] e Acar et al. [17]. Esta base foi empregada na análise e ajuste de uso dos termos que compõem o dicionário.

A segunda base, denominada de **base canvas**, é formada por 1.650 sites listados no trabalho de Englehardt et al. [61] que utilizam API Canvas para realizar *fingerprinting*. É importante ressaltar que embora o trabalho de Englehardt et al. liste mais de 16.000 sites Canvas, apenas os que possuíam mais de 100k de tamanho foram selecionados, garantindo maiores informações para análise. A terceira base, denominada **base DMOZ**, é composta por 1.584 sites extraídos aleatoriamente da diretório DMOZ, uma base composta por sites considerados benignos. Esta base foi utilizada para verificação da quantidade de termos em páginas não ligadas à *fingerprinting*.

A quarta é a **base de PhishTank** [62], é composta por 1.523 sites extraídos aleatoriamente do repositório PhishTank. Por serem considerados sites maliciosos, a intenção é verificar se esses sites podem ser considerados de *website fingerprinting*. De acordo com Nunan [63] muitos ataques de *phishing* utilizam scripts de redirecionamentos para conduzir o usuário para uma página falsa, idêntica à página original, o que pode levar o usuário a digitar credenciais de autenticação, números de cartões de crédito ou dados pessoais no site falso. A quinta e última é a **base do Alexa** [64] formada por 10.000 sites ranqueados na categoria Brasil. O ranqueamento é baseado no número estimado de visitas que o site recebe. A ideia é avaliar o níveis de *fingerprinting* em páginas brasileiras.

6.2 Resultados

Esta Seção descreve os resultados da aplicação da metodologia nas base de dados usadas na investigação da severidade de risco *fingerprinting* em páginas Web.

6.2.1 Base de Controle

A Figura 6.1 ilustra o resultado da metodologia proposta na base de controle.

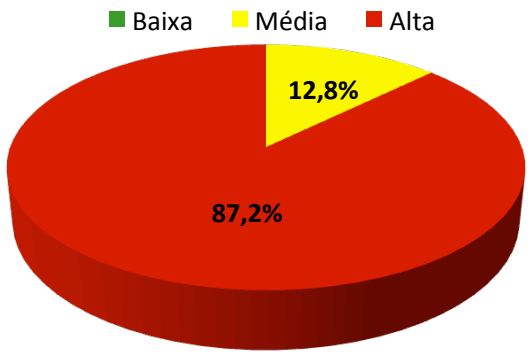

Figura 6.1: Classificação dos 250 websites que compõem a Base de Controle.

Em destaque, na cor vermelha, estão representados os sites com classificação **Alta**, perfazendo um total de (87,2%). Os sites *coinbase.com* - famoso por comprar e vender bitcoin - e *tumblr.com* - uma plataforma de blogging que permite aos usuários publicarem textos, imagens, vídeos, entre outros - possuem destaque nessa base por possuírem em seus códigos referências as quatro (4) propriedades que compõe o nível 3. Na cor média estão representados os (12,8%) dos sites classificados como **Média**.

Percebe-se que por se tratar uma base que contêm apenas páginas empregadas para *fingerprinting* não existem artefatos de baixa severidade. Isso não significa que os elementos, objetos e métodos que compõe o nível 1 não estejam presentes (na verdade, existem mais de 27.000 métodos do nível 1 nas 250 páginas da base controle), mas sim que os elementos, objetos e métodos dos outros níveis 2 e 3 estão presentes e representam um maior risco a privacidade do usuários.

A Figura 6.2 ilustra a quantidade de termos presentes nos 250 sites que compõem a base de controle em todos os níveis.

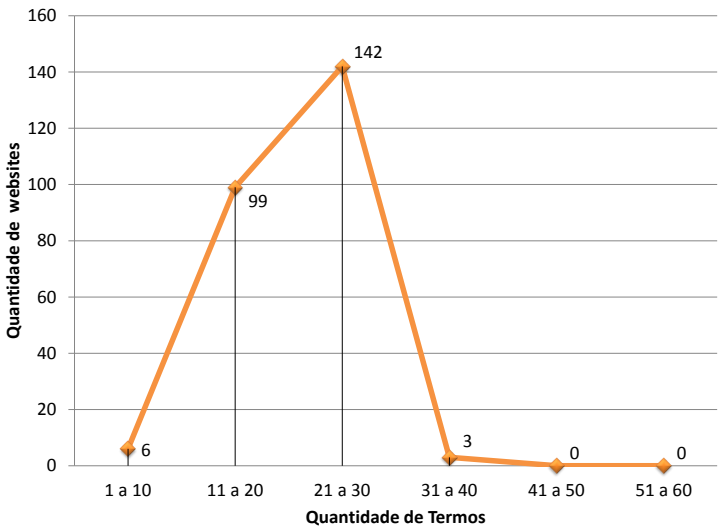

Figura 6.2: Quantidade de Termos presentes nos 250 websites da Base de Controle

Analisando os termos encontrados, pode-se afirmar que:

- Para os **6 sites** com até 10 termos não houveram ocorrências para termos do nível 3. Já os termos relacionados ao nível 2 totalizam 17 ocorrências (presença do termo no código) e 118 chamadas no código. Os termos mais prevalentes neste nível foram $referrer$, *cookie*, $mimeTypes$ e *plugins*. Por

fim, os termos relacionados ao nível 1 obtiveram 39 ocorrências e 129 chamadas no código;

- Para os **99 sites** entre 11 e 20 termos, os relacionados com o nível 3 totalizam 84 ocorrências e 384 chamadas no código, com destaque para o termo *history* com 66 ocorrências. Os termos relacionados ao nível 2 totalizam 504 ocorrências e 4.610 chamadas no código. Os termos mais prevalentes foram $referrer$, *cookie*, *javaEnabled*, *mimeTypes*, *plugins* e *localStorage*. Por fim, os termos relacionados ao nível 1 tiveram 1.081 ocorrências e 8.578 chamadas no código;

- Para os **142 sites** com maior incidência na Figura 6.2, todos os termos ligados ao nível 3 foram encontrados nos sites, com destaque para o método *history* com mais de 100 ocorrências. No total, foram encontradas 201 ocorrências e 638 chamadas dos métodos no código. Os termos ligados ao nível 2 foram: $referrer$, *cookie*, *plugins*, *mimeTypes*, *javaEnabled*() e *localStorage*, com um total de 781 ocorrências e 8.348 chamadas dos métodos no código. Por fim, os termos relacionados ao nível 1 tiveram 1.835 ocorrências e 21.553 chamadas dos métodos no código;

- Para os **03 sites** destacados com 31 a 40 termos, os termos relacionados com o nível 3, totalizam 7 ocorrências e 10 chamadas no código. Os termos relacionados ao nível 2 totalizam 21 ocorrências e 122 chamadas no código. Por fim, os termos relacionados ao nível 1 tiveram 77 ocorrências e 886 chamadas no código.

A Tabela 6.1 mostra a sumarização dos resultados, com maiores incidências para termos entre (11 a 20) e (21 a 30) no total de 241 sites com termos do Nível 1 e 2.

6.2.2 Base Canvas

A Figura 6.3 ilustra o resultado da metodologia na base Canvas.

Nesta base observa-se a ocorrência dos 3 níveis de severidade, com maior prevalência para a classificação **Alta** (65,1%) e **Média** com (34,5%), visto que a base é composta por sites com *fingerprinting* canvas. As páginas classificadas no nível 1 contabilizam seis (6) sites que ou não possuem nenhuma referência aos elementos, objetos e métodos deste nível ou, embora hajam referências, não registram termos dos níveis 2 ou 3. Isso apenas comprova que as páginas que compõem esta base têm artefatos que, de fato, possuem um nível de severidade mais elevado.

Tabela 6.1: Sumarização Quantidade de Termos - Base Controle

Nº Termos	Nº Sites	Nível 1	Nível 2	Nível 3
Ocorrências				
1 a 10	6	39	17	—
11 a 20	99	**1.081**	**504**	84
21 a 30	142	**1.835**	**781**	201
31 a 40	3	77	21	7
41 a 50	0	—	—	—
51 a 60	0	—	—	—
Chamadas				
1 a 10	6	129	118	—
11 a 20	99	**8.578**	**4.610**	384
21 a 30	142	**21.553**	**8.348**	638
31 a 40	3	886	122	10
41 a 50	0	—	—	—
51 a 60	0	—	—	—

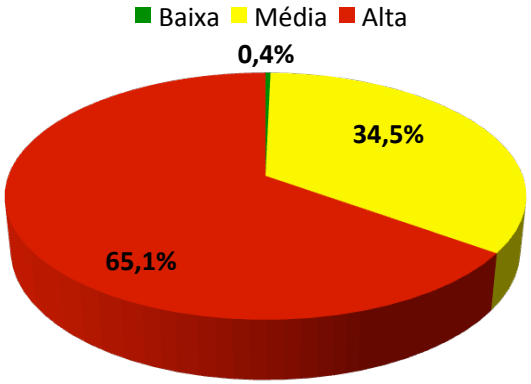

Figura 6.3: Classificação dos 1.650 websites que compõem a Base Canvas

A Figura 6.4 ilustra a quantidade de termos presentes nos 1.650 sites que compõem a base Canvas.

Analisando os termos encontrados para os 810 sites destacados na Figura 6.4, pode-se afirmar que:

1. Em relação aos termos do nível 3, foram encontrados 617 sites com ocorrências para os métodos *getImageData*() e/ou *toDataUrl*(), perfazendo (72%) do total de termos nível 3 da amostra.

2. Os termos ligados ao nível 2 mais prevalentes foram: *referrer* (696), *cookie* (736), *mimeTypes* (753), *plugins* (528). No total, foram encontradas 2.713

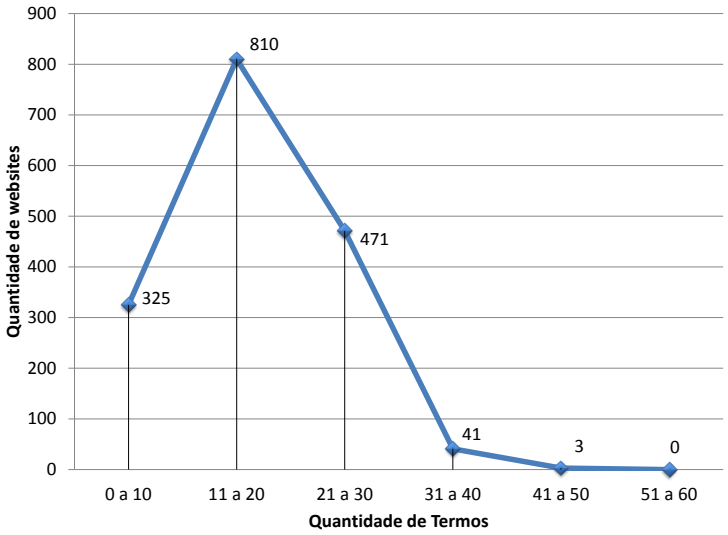

Figura 6.4: Quantidade de Termos presentes nos 1.650 webistes da Base Canvas

ocorrências e 24.116 chamadas dos métodos no código;

3. Por fim, foram encontradas 5.413 ocorrências e 49.231 chamadas dos métodos no código referentes aos termos do nível 1.

6.2.3 Base DMOZ

A Figura 6.5 ilustra o resultado da metodologia na base DMOZ.

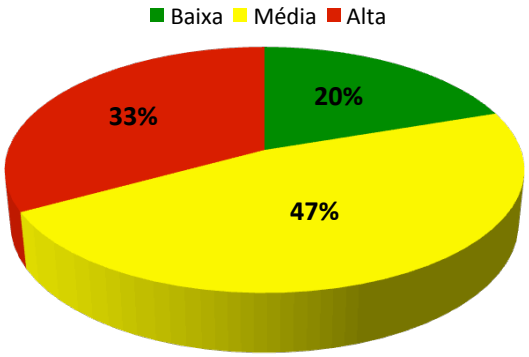

Figura 6.5: Classificação dos 1.584 websites que compõem a Base DMOZ

Nesta base observou-se a ocorrência de termos dos 3 níveis investigados. Um aspecto interessante nos resultados desta base é o fato de sites, aleatoriamente escolhidos, de um repositório usado e reconhecido como contendo páginas benignas

apresentar artefatos de todos os níveis. Percebe-se ainda claramente que alguns destes sites tem um grande potencial para invasão de privacidade.

Como forma de melhor ilustrar os resultados na base DMOZ, a Tabela 6.2 enumera as ocorrências de alguns dos termos que compõe o dicionário nesta base.

Tabela 6.2: Ocorrência de Termos encontrados nos sites da Base DMOZ

Websites	Nível 1 - Baixa					Nível 2 - Média					Nível 3 - Alta			
	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	2	2	0	0	0	0	0	0	0	0	0
joegreenephoto.com	1	5	3	1	0	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	5	5	1	0	0	0	0	0	0	0	0	0	0	0
olddogpaws.com	0	0	2	2	3	1	36	0	3	0	0	0	0	0
ilight-tech.com	9	4	8	1	1	1	2	4	8	0	0	0	0	0
bbc.com	10	7	11	32	32	52	118	6	9	7	0	0	0	0
gourmetsleuth.com	6	6	22	6	6	12	6	16	0	0	0	0	0	0
nordicacademy.com.au	4	4	1	0	0	0	3	3	5	0	0	0	0	0
nikonusa.com	29	30	37	10	12	11	49	14	58	8	0	0	0	19
flairstrips.com	14	14	17	0	3	0	12	13	3	0	0	6	1	2
robotshop.com	8	8	13	3	3	7	13	4	6	4	0	3	1	1
fibergarden.com	13	17	9	2	2	5	20	4	5	4	0	3	1	4
newscooters4less.com	5	3	0	0	0	0	0	4	0	0	0	1	1	9

Na coluna do **nível 1 - Baixa** percebe-se que os sites são usados para fornecer informações para ajustar o conteúdo ao dispositivo do usuário. Na coluna do **nível 2 - Média**, ganha destaque o site *bbc.com* apresentando todos os termos dos níveis 1 e 2, mostrando que além de obter as informações de ajustes de tela, houve a verificação dos *plugins* e *mimeTypes* existentes no navegador. Por fim, na coluna do **nível 3 - Alta** observa-se a prevalência dos termos de Canvas usados para obtenção de informações relevantes e que podem identificar o usuário unicamente na Web.

Por fim, a Figura 6.6 ilustra a quantidade de termos presentes nos 1.650 sites que compõem a base DMOZ.

Analisando os termos encontrados para os 983 sites destacados na Figura 6.6, pode-se afirmar que:

1. Apesar de ser uma base de sites benignos, foram registrados 136 sites com termos nível 3, dentre os quais *getImageData*() e/ou *toDataUrl*() com

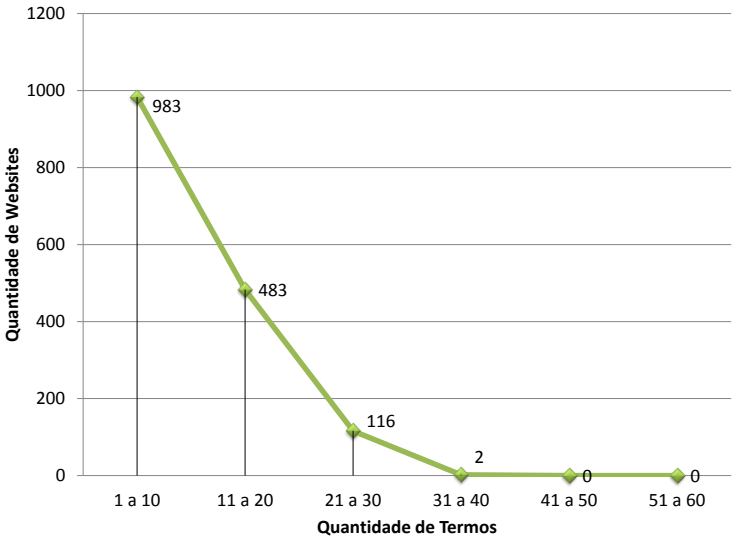

Figura 6.6: Quantidade de Termos presentes nos 1.584 webistes da Base DMOZ

mais de 70 ocorrências. No total foram encontradas 209 ocorrências e 507 chamadas dos métodos no código referentes aos quatro termos do nível 3;

2. Os termos ligados ao nível 2 foram: *referrer*, *cookie*, *mimeTypes*, *plugins* com mais de 100 ocorrências. No total, foram encontradas 1.056 ocorrências e 5.024 chamadas dos métodos no código;

3. Por fim, os termos mais prevalents ligados ao nível 1 foram: *innerWidth*, *innerHeight*, *outerWidth*, *outerHeight*, *userAgent*. No total para todos os termos nível 1 foram encontradas 2.693 ocorrências e 11.090 chamadas dos métodos no código.

6.2.4 Base PhishTank

A Figura 6.7 ilustra o resultado da metodologia proposta na base PhishTank.

Nesta base observou-se a ocorrência de termos dos 3 níveis investigados, com percentuais de 30% para classificação **Alta**, 10% para a **Baixa** e 60% para a **Média**. No total, houveram mais de 1.747 ocorrências e 18.598 chamadas nos códigos para os termos *cookie*, *plugins*, *mimeTypes* e *localStorage* que traduzem informações importantes dos usuários.

Já a Figura 6.8 ilustra a quantidade de termos presentes nos 1.524 sites que compõem a base PhishTank.

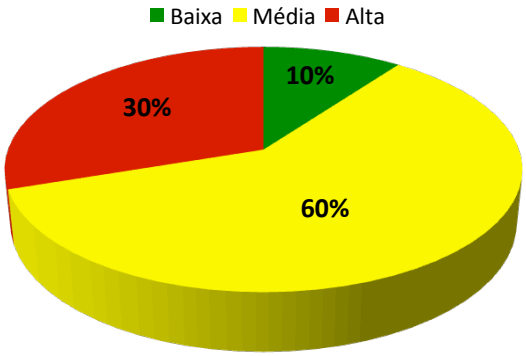

Figura 6.7: Classificação dos 1.524 websites que compõem a Base de PhishTank.

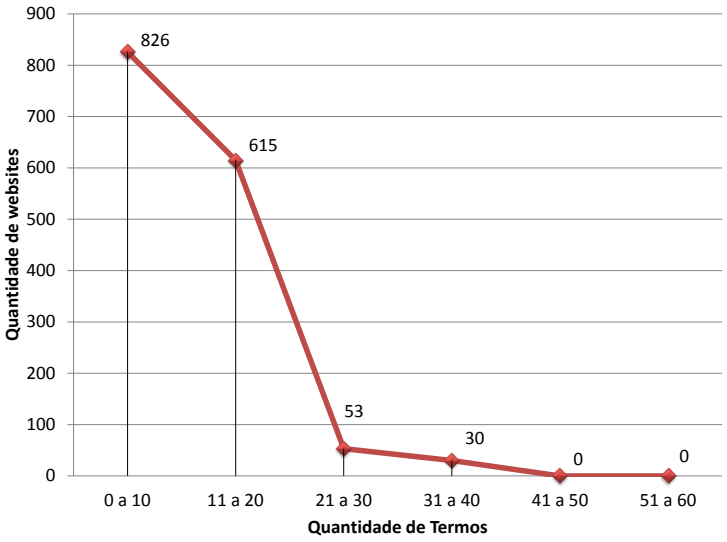

Figura 6.8: Quantidade de Termos presentes nos 1.524 webistes da Base PhishTank.

Analisando os termos encontrados para os 826 sites destacados na Figura 6.8, pode-se afirmar que:

1. A ocorrência de termos para o nível 3 foi de (30%), no total de 229 ocorrências para os termos nível 3 e 1.337 chamadas nos códigos.

2. Os termos ligados ao nível 2 foram $referrer$, *cookie*, *mimeTypes* e *plugins*, com mais de 100 ocorrências e destaque para *mimeTypes* com 499 ocorrências. No total, foram encontradas 1.098 ocorrências e 9.692 chamadas dos métodos no código;

3. Por fim, os termos mais prevalentes ligados ao nível 1 foram: *innerWidth*, *innerHeight*, *outerWidth*, *outerHeight*, *domain*, *userAgent*, *appName*, *appVersion*, *platform*, *screen.height* e *screen.width*. No total de 2.305 ocorrências e 26.350 chamadas dos métodos nos códigos.

6.2.5 Base Alexa

A Figura 6.9 ilustra o resultado da metodologia proposta na base Alexa.

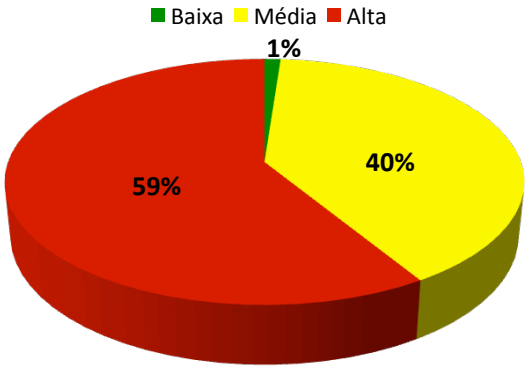

Figura 6.9: Classificação dos 10.000 websites que compõem a Base Alexa.

Nesta base observou-se a ocorrência de termos dos 3 níveis investigados. Na classificação **Alta**, o percentual foi de 59% (com 10.025 ocorrências e 28.026 chamadas no código). A classificação **Média** alcançou 40% (com 27.074 ocorrências e 141.260 chamadas no código) e a **Baixa** com 1% (com 261 ocorrências e 616 chamadas no código).

Já a Figura 6.10 ilustra a quantidade de termos presentes nos 10.000 sites que compõem a base Alexa.

Analisando os termos encontrados para os 5.172 sites destacados na Figura 6.10, pode-se afirmar que:

1. A ocorrência para os termos nível 3 é de 5.903 ocorrências com 16.360 chamadas nos códigos, com destaque para o termo *history* com 2.514 ocorrências, seguidos por *toDataURL*() (1.730) e *getImageDat*()*a* (1.648);

2. Para termos nível 2 houveram 22.157 ocorrências e 113.371 chamadas nos códigos, com destaque para os termos *referrer*, *cookie*, *plugins*, *Modernizr*, *javaEnabled* e *mimeTypes* com mais de 1.000 ocorrências cada.

3. Por fim, para os termos que compõe o nível 1 houveram 49.826 ocorrências e 350.292 chamadas no código.

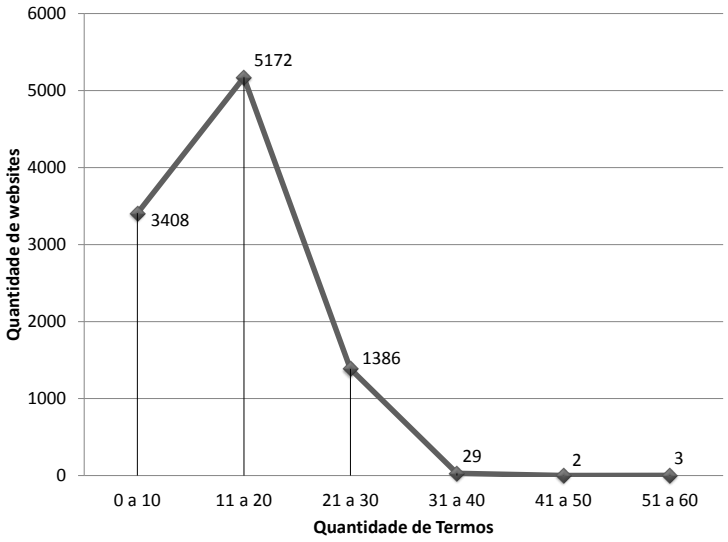

Figura 6.10: Quantidade de Termos presentes nos 10.000 websites da Base Alexa

6.3 Discussão

A Tabela 6.3 apresenta, de forma resumida, uma comparação entre os resultados obtidos nas 5 (cinco) base de dados, ilustrando os percentuais obtidos em cada uma.

Tabela 6.3: Percentuais de classificação das Bases de Dados

Base	Nº Sites	Baixa	Média	Alta
Controle	250	0%	12,8%	**87,2%**
Canvas	1.650	0,4%	34,5%	**65,1%**
DMOZ	1.584	20%	**47%**	33%
PhishTank	1.523	10%	**60%**	30%
Alexa	10.000	1%	40%	**59%**

As bases que obtiveram as maiores classificações de sites com severidade **Alta** foram a de Controle, a Canvas e a Alexa.com, com **87,2%**, **65,1%** e **59%**, respectivamente. A explicação para tal fato em relação as duas primeiras bases (Controle e Canvas) é até certo ponto simples, visto que ambas são compostas realmente por sites relacionados (rotulados) com *fingerprinting*. O resultado "não esperado" foi o da base Alexa.com, visto que ela é composta por sites hospedados no Brasil ligados a atividades como e-commerce, entretenimento,

educação, noticiais, entre outros. Um bom exemplo disso é o site *dell.com.br*, que possui 44 dos 58 termos analisados. Foram encontrados 34 termos do nível 1, os termos *cookie*, *Modernizr*, *Silverlight*, *WebGLRenderingContext*, *cookieEnabled*, *javaEnabled*, *mimeTypes*, *plugins* e *localStorage* do nível 2 e o termo *history* do nível 3. Isso nos permite afirmar que o site realmente faz uma coleta de informações que podem comprometer a privacidade do usuário.

As duas últimas bases (DMOZ e PhishTank) obtiveram as maiores classificações de sites com severidade **Média**, com **47%** e **60%**, respectivamente. Um aspecto observado nestas bases é a ocorrência de poucos termos do nível 2. Dos 10 termos ligados ao nível, três tem maior prevalência (*cookie*, *plugins* e *mimeTypes*). Esse resultado, mostra que nem mesmo sites considerados benignos estão livres de realizar *fingerprinting*. Já o uso em sites maliciosos só vem comprovar que os atacantes precisam e utilizam essas informações para, por exemplo, conduzir o usuário a uma página falsa, idêntica à página original, e assim realizar o roubo de informações (credenciais de autenticação, números de cartões de crédito ou dados pessoais) privadas.

Mudando o foco dos níveis para os termos, a análise das bases mostrou a presença de todos os 58 termos utilizados, com maior prevalência para aqueles ligados ao ajuste de tela e conteúdo (*innerWidth*, *innerHeigth*, *outerWidth*, *outerHeigth*, *screen.Height* e *screen.Width*) e dados do navegador (*userAgent*). Já os termos ligados ao nível 2, o que mais se destacam são *referrer*, *cookie*, *mimeTypes* e *plugins*, empregados para revelar informações mais detalhadas do navegador/dispositivo como, por exemplo, a lista de plugins instalados (onde pode ser visto que o usuário possui algum softwares específico ligado à *internet banking*). A Tabela 6.4 ilustra a quantidade de termos encontradas nas bases.

Tabela 6.4: Ocorrência e Chamadas dos Termos encontrados nas Bases

	Nível 1 - Baixa					Nível 2 - Média					Nível 3 - Alta			
Bases	innerWidth	innerHeight	userAgent	domain	outerWidth	screen.width	referrer	cookie	mimeTypes	plugins	getUserMedia	getImageData	toDataURL()	history
Ocorrências														
Controle	239	238	246	232	—	—	225	242	231	231	3	40	43	206
Canvas	1387	1389	1664	—	—	1322	1473	1478	1524	1095	4	499	598	774
DMOZ	802	786	1097	—	—	634	654	846	811	644	0	258	273	359
PhishTank	917	943	1194	769	—	—	633	862	1067	707	2	270	274	287
Alexa	8225	7990	9337	—	7364	—	5686	7158	8546	5805	18	2890	3063	4054
Chamadas														
Controle	1878	2629	2707	3648	—	—	2106	5681	1745	2101	4	97	60	835
Canvas	11420	11959	18980	—	—	5646	12507	24337	11200	9213	9	1865	932	4069
DMOZ	4201	4524	6836	—	—	1674	2922	8441	4320	3175	0	911	319	1241
PhishTank	9652	10490	13168	6296	—	—	5273	13909	10718	7313	13	2057	731	1902
Alexa	53486	54813	63568	—	133635	—	23765	75542	43808	33161	56	10791	4069	13110

Para finalizar, é possível afirmar que o objetivo dessa dissertação (uma abordagem para mensurar a severidade de um artefato *fingerprirnitng* presente em uma página Web), foi alcançado, uma vez que os resultados em base de sites reais provam e comprovam a presença de termos *fingerprinting* em diferentes níveis.

Capítulo 7

Considerações Finais

As técnicas de *fingerprinting* aplicadas a páginas Web estão se tornando cada vez mais presentes e comuns, mesmo assim os usuários não são informados sobre isso. Embora as formas, os mecanismos e suas consequências sejam conhecidos, questões como a privacidade dos usuários e as formas de evitar a coleta de informações ainda são motivo de discussão. A existência de uma indústria de propaganda especializada, evoluída e financeiramente feliz, aliada ao surgimento de novos serviços e tecnologias, assim como, a constante evolução tecnológica e ao crescente número de usuários (em atividades que envolvem dados pessoais e valores) impõe novos desafios na atividade de detectar e mitigar o *fingerprinting*.

Este trabalho propôs uma metodologia para detectar artefatos (scripts) de *fingerprinting* em páginas Web e informar aos usuários o quão perigosa a página é para sua privacidade. Para tanto, o conjunto de propriedades e métodos dos objetos HTML DOM foram avaliados e o nível de risco à privacidade dos usuários foi estimado.

Como forma de prová-la, várias bases de páginas Web contendo ou não *fingerprinting*, foram testadas e o resultado mostra que os artefatos para este fim são comuns e presentes em grande parte das páginas na Internet.

7.1 Dificuldades encontradas

Com o intuito de detectar *fingerprinting* em páginas Web, inicialmente foi pensada a utilização de grafo de dependência[1]. A ideia era criar um base de grafos dos scripts de *fingerprinting* que seriam comparados com os script extraídos de páginas Web em tempo real. Contudo, durante a pesquisa percebeu-se que não seria possível analisar as propriedades *fingerprinting* por meio do grafo de dependên-

[1]De acordo com Martins et al. [65], grafos de dependência são grafos direcionados que representam relações de dependência entre elementos pertencentes a uma mesma estrutura.

cia, pois a principal finalidade do trabalho é detectar o uso de uma propriedade específica e não como a estrutura está disposta no código.

Outra ideia, foi fazer uso de recuperação de informação, mas especificamente do Modelo Vetorial[2]. A ideia é que cada documento (página Web) fosse representado como um vetor de termos e cada termo possuísse um valor associado que indicasse o grau de importância (peso) deste termo em determinado documento. O ranqueamento dos scripts de *fingerprinting* foi analisado com o classificador k-NN. Contudo, nos resultados percebeu-se que todos os sites possuem algumas das propriedades de *fingerprinting*. Assim, sites rotulados como não *fingerprinting* foram ranqueados como *fingerprinting*.

Foi neste ponto da pesquisa que percebeu-se que praticamente todos os sites possuem alguma propriedade de *fingerprinting*, sendo usada para fins benéficos ou para atividades maliciosas. Assim, surgiu a ideia de classificar os sites pelo nível de severidade de acordo com o nível de risco que as propriedades detectadas nos scripts podem causar ao usuário.

7.2 Contribuições Alcançadas

Esta dissertação contribui com:

1. A criação de uma abordagem para medir a severidade (risco de ameaça à privacidade) de artefatos *fingerprint* em páginas Web;

2. Lista de Termos classificados por nível de risco;

Além disso, esta dissertação incentivou a criação de um mecanismo para *website fingerprinting*, que serviu de projeto final de graduação e foi recentemente aprovada no X Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG) do SBSeg 2016. Também foi fruto desta dissertação um minicurso ("Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas"), publicado no livro de minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2014). Por fim, também fruto deste trabalho, o artigo "Determinando o Risco de Fingerprinting em Páginas Web", foi aceito no XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2016).

[2]É um modelo onde documentos e consultas são vistos como característica num espaço vetorial n-dimensional, sendo a distância vetorial usada como medida de similaridade

7.3 Trabalhos Futuros

As pesquisas sobre *fingerprinting* estão sendo atualizadas constantemente, pois, novos experimentos surgem, trazendo para a comunidade científica inovações que aprimoram processos focados nessa área. Os trabalhos futuros estão elencados a seguir:

1. Classificar os websites *fingerprinting* quanto ao nível de severidade de risco utilizando aprendizagem de máquina;
2. Utilizar técnicas de entropia para verificar o termo ou um conjunto de termos que possuam uma maior relevância para a detecção de *fingeprinting*;
3. Manter e categorizar uma base de scripts para ajudar a comunidade acadêmica nos estudos referentes aos *fingerprinting*.
4. Mecanismo para verificar ofuscação de termos *fingerprinting* em sites Web;

Referências Bibliográficas

[1] D. Kristol and L. Montulli, "Http state management mechanism." RFC 2109 (Proposed Standard), 1997. http://www.ietf.org/rfc/rfc2109.txt.

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

[3] J. R. Mayer, *Any person... a pamphleteer: Internet Anonymity in the Age of Web 2.0.* PhD thesis, Princeton University, 2009.

[4] A. Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith, "Privacy Considerations for Internet Protocols." RFC 6973 (Informational), July 2013. http://www.ietf.org/rfc/rfc6973.txt.

[5] P. Giura, I. Murynets, R. Piqueras Jover, and Y. Vahlis, "Is it really you?: User identification via adaptive behavior fingerprinting," in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14, (New York, NY, USA), pp. 333–344, ACM, 2014.

[6] D. Goodin, "Stealthy technique fingerprints smartphones by measuring users movements." http://arstechnica.com/security/2013/10/stealthy-technique-fingerprints-smartphones-by-measuring-users-movements/, 2013.

[7] J. Forshaw, "WebGL - A New Dimension for Browser Exploitation." http://www.contextis.com/resources/blog/webgl-new-dimension-browser-exploitation/, 2011.

[8] Contextis, "WebGL - More WebGL Security Flaws." http://www.contextis.com/resources/blog/webgl-more-webgl-security-flaws/, 2011.

[9] W3C, "Fingerprinting guidance for web specification authors," 2014. http://w3c.github.io/fingerprinting-guidance/.

[10] A. Barth, "HTTP State Management Mechanism." RFC 6265 (Proposed Standard), apr 2011. http://www.ietf.org/rfc/rfc6265.txt.

[11] S. Kamkar, "Evercookie." http://samy.pl/evercookie/, 2010.

[12] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, (Washington, DC, USA), pp. 541–555, IEEE Computer Society, 2013.

[13] T. Unger, M. Mulazzani, D. Fruhwirt, M. Huber, S. Schrittwieser, and E. Weippl, "Shpf: Enhancing http(s) session security with browser fingerprinting," in *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pp. 255–261, Sept 2013.

[14] M. Mulazzani, M. Huber, M. Leithner, and S. Schrittwieser, "Fast and reliable browser identification with javascript engine fingerprinting," in *Web 2.0 Workshop on Security and Privacy*, (W2SP), 2013.

[15] StatCounter, "Statcounter globalstats." http://gs.statcounter.com, 2014.

[16] W3C, "Document Object Model (DOM)." https://www.w3.org/DOM/, 2016.

[17] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "Fpdetective: Dusting the web for fingerprinters," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, (New York, NY, USA), pp. 1129–1140, ACM, 2013.

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." RFC 2616 (Draft Standard), June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[19] W3C, "Html5: a vocabulary and associated apis for html and xhtml." http://www.w3.org/TR/html5/, 2014. [Online, Acessado 15/08/2014].

[20] S. Pemberton, "Xhtml 1.0: The extensible hypertext markup language (second edition)." World Wide Web Consortium, Recommendation REC-xhtml1-20020801, August 2002.

[21] A. L. Hors, P. L. Hégaret, and J. Stenback, "Document object model (DOM) level 2 HTML specification," W3C recommendation, W3C, Jan. 2003. http://www.w3.org/TR/2003/REC-DOM-Level-2-HTML-20030109.

[22] S. Fulton and J. Fulton, *HTML5 Canvas - Native Interactivity and Animation for the Web.* O'Reilly, 2011.

[23] A. R. Slivinski, "Desenvolvimento de uma Metodologia para Avaliação de Desempenho Gráfico 3D de Plataformas com Suporte ao WebGL," tech. rep., Unioeste - Universidade Estadual do Oeste do Paraná, 2011.

[24] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," in *Proceedings of W2SP 2012* (M. Fredrikson, ed.), IEEE Computer Society, May 2012.

[25] J. Kirk, "Canvas fingerprinting online tracking is sneaky but easy to halt." http://www.pcworld.com/article/2458280/canvas-fingerprinting-tracking-is-sneaky-but-easy-to-halt.html, 2014.

[26] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *21st ACM Conference on Computer and Communications Security*, ACM CCS 2014, Nov. 2014.

[27] D. Crockford, *JavaScript - the good parts: unearthing the excellence in JavaScript.* O'Reilly, 2008.

[28] ECMA International, *Standard ECMA-262 - ECMAScript Language Specification.* 5.1 ed., June 2011. http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[29] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, "Fingerprinting information in JavaScript implementations," in *Proceedings of Web 2.0 Security and Privacy 2011 (W2SP)*, (San Franciso), May 2011.

[30] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle, "Flash cookies and privacy.," in *AAAI Spring Symposium: Intelligent Information Privacy Management*, 2010.

[31] A. M. McDonald and L. F. Cranor, "A survey of the use of adobe flash local share objects to respawn http cookies," *Journal of Information Policy*, vol. 7, no. 3, pp. 639–687, 2011.

[32] K. Group, "Webgl 2 specification," khronos draft, Khronos Group, 2014. https://www.khronos.org/registry/webgl/specs/latest/2.0/.

[33] BrowserLeaks.com, "Web browser security ? browserleaks.com," 2015. https://www.browserleaks.com.

[34] S. Shankland, "Microsoft declares webgl 'harmful' to security." http://www.cnet.com/news/microsoft-declares-webgl-harmful-to-security/, 2011.

[35] B. Bos, T. Celik, I. Hickson, and H. W. Lie, "Cascading style sheets level 2 revision 1 (css 2.1) specification," w3c recommendation, W3C, July 2011. http://www.w3.org/TR/CCS2.

[36] Jeremiah Grossman, "I know where you've been," 2006. http://jeremiahgrossman.blogspot.com.br/2006/08/i-know-where-youve-been.html.

[37] Microsoft, "Microsoft silverlight 5." http://www.microsoft.com/silverlight/, 2014.

[38] P. Eckersley, "How unique is your web browser?," in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, (Berlin, Heidelberg), pp. 1–18, Springer-Verlag, 2010.

[39] K. Boda, A. M. Földes, G. G. Gulyás, and S. Imre, "User tracking on the web via cross-browser fingerprinting," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7161 LNCS, pp. 31–46, 2012.

[40] T. Yen, Y. Xie, F. Yu, R. Yu, and M. Abadi, "Host fingerprinting and tracking on the web: Privacy and security implications," *... Distributed System Security ...*, 2012.

[41] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An empirical study of privacy-violating information flows in JavaScript web applications," *Proceedings of the 17th ACM conference on Computer and communications security - CCS '10*, p. 270, 2010.

[42] L. Olejnik, C. Castelluccia, and A. Janc, "On the uniqueness of Web browsing history patterns," *Annals of Telecommunications - Annales Des Télécommunications*, vol. 69, pp. 63–74, Sept. 2013.

[43] J. Yu and E. Chan-Tin, "Identifying Webbrowsers in Encrypted Communications," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society - WPES '14*, (New York, New York, USA), pp. 135–138, ACM Press, 2014.

[44] J. Mayer, A. Narayanan, and S. Stamm, "Do not track: A universal third-party web tracking opt out." IETF Draft, 2011.

[45] W3C, "Tracking preference expression (dnt)," w3c working draft, W3C, April 2014. http://www.w3.org/TR/tracking-dnt/.

[46] Eletronic Frontier Foundation, "Do not track," 2014. https://www.eff.org/issues/do-not-track.

[47] F. Future of Privacy Forum, "About do not track." http://allaboutdnt.com, 2014.

[48] Tor Project, "The design and implementation of the tor browser," 2014. https://www.torproject.org/projects/torbrowser/design/.

[49] T. Wang and I. Goldberg, "Improved website fingerprinting on Tor," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society - WPES '13*, (New York, New York, USA), pp. 201–212, ACM Press, 2013.

[50] M. Perry, E. Clark, and S. Murdoch, "The design and implementation of the tor browser [draft]." https://www.torproject.org/projects/torbrowser/design/, may 2013.

[51] FireGloves, "Firegloves - cross-browser fingerprinting test 2.0." http://fingerprint.pet-portal.eu/?menu=6, 2014.

[52] Evidon, "Ghostery." http://www.ghostery.com/, 2014.

[53] Adblock Plus, "Adblock plus surf the web without annoying ads." https://adblockplus.org/, 2014.

[54] INRIA, "Stopfingerprinting - how trackable is your browser in a long term?." https://stopfingerprinting.inria.fr, 2014.

[55] Mozilla, "Lightbeam shine a light on whos watching you." https://www.mozilla.org/en-US/lightbeam/, 2014.

[56] 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*, WWW '15, (New York, NY, USA), pp. 820–830, ACM, 2015.

[57] A. F. Khademi, "Browser fingerprinting: Analysis, detection, and prevention at runtime," Master's thesis, Queen's University, 2014.

[58] T. A. Sudkamp, *Languages and Machines: An Introduction to the Theory of Computer Science (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[59] H. Tillmann, "Browser fingerprinting: Tracking ohne spuren zu hinterlassen," Master's thesis, Humboldt-UniversitÃ¤t zu Berlin, 2013.

[60] Y. Tian, Y. C. Liu, A. Bhosale, L.-S. Huang, P. Tague, and C. Jackson, "All your screens are belong to us: Attacks exploiting the html5 screen sharing api," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pp. 34–48, 2014.

[61] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," 2016.

[62] PhishTank, "Freencommunity site for anti-phishing service." http://http://www.phishtank.com/, 2016.

[63] A. E. Nunan, "Detecção de cross-site scripting em páginas web," Master's thesis, 2012. Instituto de Computação.

[64] Amazon, "Alexa." http://http://www.alexa.com/, 2016.

[65] G. B. Martins, E. Souto, R. Freitas, and E. Feitosa, "Estruturas virtuais e diferenciaÃ§Ã£o de vertices em grafos de dependencia para deteccao de malware metamorfico," in *XIV Simposio Brasileiro de Seguranca da InformacÃ o e de Sistemas Computacionais (SBSeg14)*, (Belo Horizonte, MG, Brasil), pp. 260–273, SBC, 2014.

Apêndice A

Dicionário de Termos

Listagem A.1: Pseudocódigo para Extração de Termos

```
1;window;innerWidth
1;window;innerHeight
1;window;outerWidth
1;window;outerHeight
1;window;devicePixelRatio
1;document;domain
1;navigator;userAgent
1;navigator;appCodeName
1;navigator;appName
1;navigator;appVersion
1;navigator;appMinorVersion
1;navigator;buildID
1;navigator;browserLanguage
1;navigator;cpuClass
1;navigator;doNotTrack
1;navigator;language
1;navigator;onLine
1;navigator;oscpu
1;navigator;platform
1;navigator;userLanguage
1;navigator;product
1;navigator;productSub
1;navigator;securityPolicy
1;navigator;systemLanguage
1;navigator;vendor
1;navigator;vendorSub
1;navigator;navigator.geolocation
1;navigator;savePreferences
1;screen;availHeight
1;screen;availWidth
1;screen;availLeft
1;screen;availTop
1;screen;screen.height
1;screen;screen.width
1;screen;colorDepth
1;screen;pixelDepth
1;screen;bufferDepth
1;screen;deviceXDPI
1;screen;deviceYDPI
1;screen;logicalXDPI
```

```
1;screen;logicalYDPI
1;screen;systemXDPI
1;screen;systemYDPI
1;screen;updateInterval
2;document;referrer
2;document;cookie
2;Modernizr;Modernizr
2;Silverlight;Silverlight
2;WebGL;WebGLRenderingContext
2;navigator;cookieEnabled
2;navigator;javaEnabled
2;mimeTypes;mimeTypes
2;plugins;plugins
2;window;localStorage
3;navigator;getUserMedia
3;canvas;toDataURL
3;canvas;getImageData
3;window;history
```

Apêndice B

Propriedades e Métodos Fingerprinting

Tabela B.1: Propriedades e Métodos do Objeto Window

Objeto Window (window.property)			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
innerWidth	Retorna a largura interna de uma janela de área de conteúdo.	innerHeight	Retorna a altura interna de uma janela de área de conteúdo.
outerWidth	Retorna a largura exterior de uma janela, incluindo todos os elementos da interface (como barras de ferramentas / barras de rolagem).	outerHeight	Retorna a altura exterior de uma janela, incluindo todos os elementos da interface (como barras de ferramentas / barras de rolagem).
history	Contém os URLs visitados pelo usuário	localStorage	Armazenamento local, melhor do que cookies.
silverlight	usado para identificar a presença ou não do Microsoft Silverligth no computador.	devicePixelRatio	propriedade somente de leitura retorna a razão do tamanho (vertical) de um pixel físico no dispositivo.
WebGLRenderingContext	Fornece métodos, propriedades e constantes que permitem criar desenhos em 2D e gráficos 3D WebGL.		

Tabela B.2: Propriedades e Métodos do Objeto Document

Propriedades do Objeto Document (document.property)			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
referrer	retorna uma String, que representa o URL do documento que carregou o documento atual.	getElementById ().	possibilita acessar um elemento <canvas> usando getElementById('canvas')
domain	retorna uma String, que representa o nome de domínio do servidor que carregou o documento atual, ou null se o domínio do documento não pode ser identificado.	getElementsByTagName ()	pode-se criar um elemento <canvas> usando o método document.getElementsByTagName('canvas')
cookie	Os cookies permitem armazenar informações do usuário em páginas da web.	document.createElement ()	pode-se criar um elemento <canvas> usando o método createelement('canvas').

Tabela B.3: Propriedades e Métodos do Objeto Navigator

Propriedades do Objeto Navigator (navigator.property)			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
userAgent	String com dados do cabeçalho http, uma combinação das propriedades appCodeName e appVersion.	doNotTrack	Configuração do-not-track do usuário. Este é "sim"se o usuário, solicitou para não ser rastreados por sites, conteúdo ou publicidade.
appCodeName	Nome fantasia do navegador	cpuClass	String com a classe da (CPU) do SO.
appVersion	Versão do navegador e sistema operacional	appMinorVersion	Informações adicionais sobre a versão do navegador.
appName	Nome do navegador	language	Idioma do navegador.
geolocation	Localização geográfica do utilizador.	mimetype	Retorna um objeto MimeTypeArray, que contém uma lista de objetos que representam os tipos MIME reconhecidos pelo browser.
buildID	Retorna o identificador de configuração do browser.	onLine	Valor booleano que especifica se o navegador está em modo online ou offline.
browserLanguage	Idioma do navegador	plugins	Retorna um objeto PluginArray, listando os plug-ins instalados na aplicação.
cookieEnabled	Verifica se o navegador pode ler ou escrever cookies	userLanguage	Idioma de preferência do usuário, geralmente o idioma da interface do usuário.
oscpu	Retorna o sistema operacional atual	securityPolicy	Contém as definições de política de segurança atual.
platform	Retorna o sistema operacional	systemLanguage	Idioma padrão usado pelo SO.
product	Retorna o nome do mecanismo do navegador	vendorSub	Retorna número de versão do fornecedor.
productSub	Informações adicionais sobre o mecanismo do navegador.	vendor	Retorna o nome do fornecedor do navegador.
Métodos do Objeto Navigator: navigator.método( )			
Método	**Descrição**	**Método**	**Descrição**
savePreferences	Salva os valores de preferência para o usuário atual.	javaEnabled	Verifica se o navegador suporta ou não Java
getStorageUpdates	Bloqueia um script, assim os valores de cookies e itens nos objetos de armazenamento de atributos LocalStorage podem mudar.	getUserMedia	Solicita permissão para utilizar um dispositivo multimídia,como uma câmera ou um microfone.
Propriedades navigator.mimetypes [índice].propriedade			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
name	Tipo MIME no formato tipo/subtipo.	description	Descrição do tipo de conteúdo representado pelo MIME.
suffixes	A string que lista as possíveis extensões de arquivo para o tipo MIME, por exemplo "mpeg, mpg, mpe, mpv, vbs, mpegv".	enabledPlugin	Referência ao Plugin objeto configurado para o tipo MIME.
length	Quantidade de MIME associados ao plugin.	type	O nome do tipo de MIME, por exemplo "video/mpeg"ou "audio/x-wav".
Propriedades navigator.plugins.propriedade [índice].propriedade			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
name	Retorna o nome do plugin.	filename	Retorna o nome do arquivo do plugin.
description	Retorna a descrição do plugin.	length	Quantidade de tipos MIME suportados. Mesmo que mimetypes.length.
version	Retorna a sequência de número de versão do plugin		

Tabela B.4: Propriedades e Métodos do Objeto Screen

Propriedades do Objeto Screen (screen.property)			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
availHeight	Retorna a altura da tela do usuário.	availWidth	retorna a largura da tela do usuário.
availLeft	Retorna o lado esquerdo da área na tela que está disponível para janelas de aplicativos.	availTop	Retorna o lado superior da área da tela que está disponível para janelas de aplicativos.
height	Retorna a resolução vertical da tela de exibição, em pixels.	width	Largura total da tela em pixels.
colorDepth	Recupera o número de bits usados para representar a cor de um único pixel na tela ou no buffer quando o buffer off-screen é permitido.	pixelDepth	Recupera o número de bits usados para representar a cor de um único pixel na tela ou no buffer quando o buffer off-screen é permitido.
bufferDepth	Define ou recupera o número de bits utilizados para representar a cor de um pixel único no buffer de bitmap fora da tela.	deviceXDPI	Retorna o número atual de pontos por polegada (DPI) da janela de exibição do documento ao longo do eixo (x) horizontal.
deviceYDPI	Retorna o número atual de pontos por polegada (DPI) da janela de exibição do documento ao longo da (y) eixo vertical.	logicalXDPI	Retorna o número de pontos por polegada (DPI) da janela de exibição do documento ao longo do eixo (x) horizontal no nível de zoom normal.
logicalYDPI	Retorna o número de pontos por polegada (DPI) da janela de exibição do documento ao longo da (y) eixo vertical a nível de zoom normal.	systemXDPI	Retorna o número de pontos por polegada (DPI) da tela de exibição ao longo do eixo (x) horizontal no nível de zoom normal.
systemYDPI	Retorna o número de pontos por polegada (DPI) da tela de exibição ao longo do eixo (y) horizontal no nível de zoom normal.	updateInterval	Específica ou retorna o intervalo de tempo (em milissegundos) entre as atualizações de tela.

Tabela B.5: Microsoft Silverligth

Propriedades do Objeto Window (window.propriedade)			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
isInstalled	Indica se o versão especificada do Silverlight está instalado.	IsVersionSupported	Retorna true se o plug-in Silverlight existente ou é compatível com a cadeia de versão especificada, caso contrário, false.
WaitForInstallCompletion	verifica periodicamente para novas instalações Silverlight e atualiza automaticamente a janela do navegador após a instalação.	isBrowserRestartRequired	Indica se o plug-in Silverlight já está instalado quando os carrega páginas da web, que requerem uma reinicialização do navegador se o plug-in é atualizado.

Tabela B.6: Biblioteca Modernizr

Propriedades da Biblioteca Modernizr			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
geolocation	Detecta suporte para a API de Geolocalização para os usuários a fornecer a sua localização para aplicações web.	applicationcache	Detecta o suporte para o cache de aplicativos, para armazenamento de dados para permitir que aplicações baseadas na web possam executar offline.
canvas	Detecta o suporte para o elemento <canvas> para o desenho 2D.	cookies	Detecta se o suporte a cookies está ativada.
flash	Detecta suporte a Flash, bem como plugins Flash bloqueados.	history	Detecta suporte para a API history para manipular o histórico de sessões do navegador.
localstorage	Detecta suporte para o armazenamento local e não tem limite de tempo para que os dados devem sejam mantidos.	sessionstorage	Detecta ssuporte ao armazenamento de sessão, por outro lado (como o nome sugere) armazena os dados para apenas uma sessão.
webglextensions	Detecta o suporte para extensões OpenGL em WebGL. Se o API extensões WebGL é suportado, em seguida, expõe as extensões suportadas como subpropriedades.		

Tabela B.7: Biblioteca WebGL

Propriedades do WebGL			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
RENDERER	Retorna o nome do RENDERER. Este nome é tipicamente específica para uma configuração particular de uma plataforma de hardware.	VENDOR	Retorna o nome do representante. Este nome é tipicamente específica para uma configuração particular de uma plataforma de hardware. Ela não muda de versão para versão.
VERSION	Retorna uma versão ou número de release do WebGL.	SHADING LANGUAGE VERSION	Retorna uma versão ou release número do formulário WebGL <espaço> GLSL <espaço> é <espaço> 1,0 <espaço> <informações específicas do fornecedor>.

Tabela B.8: HTML Canvas

Propriedades do Objeto Window (window.propriedade)			
Propriedade	**Descrição**	**Propriedade**	**Descrição**
getContext	O método getContext () retorna um objeto que fornece métodos e propriedades para desenhar na tela.	getImageData	Retorna um objeto ImageData que copia os dados de pixel para o retângulo especificado com o canvas.
fillRect	Desenha um retângulo "preenchido".	fillText	Desenha um texto.
fillStyle	Define ou retorna a cor, gradiente ou padrão usado para preencher o desenho	font	Define ou retorna as propriedades da fonte de corrente para o conteúdo do texto.
textBaseline	Define ou retorna a linha de base do texto atual usado na elaboração do texto.	toDataURL	O método toDataURL () do elemento de tela permite especificar o formato no qual deseja devolver o url dados. Por padrão, ele retorna image/png, mas especificando image/jpeg deve retornar uma imagem JPEG.
width	Retorna a largura de um objeto ImageData.	height	Retorna a altura de um objeto ImageData.
strokeStyle	Executa um conjunto de propriedades como cor, gradiente e padrões para um caminho a ser desenhado.	lineTo	Adiciona um novo ponto e cria uma linha a partir do último ponto especificado na tela.