Capítulo 2

Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas

Adriana Rodrigues Saraiva, Pablo Augusto da Paz Elleres, Guilherme de Brito Carneiro e Eduardo Luzeiro Feitosa

Abstract

Fingerprinting methods are those used to identify (or re-identify) a user or a device through a set of attributes (device screen size, versions of installed software, and so on) and other observable features during communication process. Commonly known as Device Fingerprinting, such techniques can be used as a security measure (to authenticate users, for example) and as mechanism for sales/marketing. Unfortunately, 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. This Chapter introduces the concepts and techniques of device fingerprinting that allow to obtain data on users and their devices. Also presents tools and countermeasures to deal with the problem.

Resumo

Técnicas de fingerprinting são aquelas empregadas para identificar (ou reidentificar) 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 processo de comunicação. Comumente conhecidas por Device Fingerprinting, tais técnicas podem ser usadas como medida de segurança (na autenticação de usuários, por exemplo) e como mecanismo para vendas/marketing. Infelizmente, também podem ser consideradas uma ameaça potencial a 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. Este Capítulo apresenta os conceitos e as técnicas de device fingerprinting que permitem obter dados relativos aos usuários e seus dispositivos. Apresenta também ferramentas e contramedidas para lidar com o problema.

2.1. Introdução

A Internet se tornou uma realidade na vida de milhões de pessoas. 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 as informações e na execução de tarefas. Entretanto, esses mesmos avanços acabaram ou permitiram a manipulação de uma das características básicas da origem da Internet, o anonimato. O simples ato de navegar e acessar sites e serviços de forma anônima, ou através do uso de pseudônimos, já não é mais verdade.

O trabalho de [Nikiforakis et al. 2014], entitulado "Browser Fingerprinting and the on-line-Tracking Arms Race" relata um fato de julho de 1993, onde uma charge de Peter Steiner, publicada pelo jornal The New Yorker, mostrava um labrador sentado em uma cadeira na frente de um computador, digitando e comentando para um companheiro (um beagle) ao lado: "Na Internet, ninguém sabe que você é um cachorro". Passadas duas décadas, as partes interessadas na comunicação do cachorro, não só saberiam que ele é um cão, como teriam uma boa ideia da sua cor, de quantas vezes ele foi ao veterinário e qual o seu biscoito favorito. O fato é que hoje é muito fácil identificar um usuário na Internet.

A premissa básica é que o ato de navegar deixa vários rastros. Por exemplo, ao acessar um site, um usuário tem o seu dispositivo (computador, tablet, smartphone, entre outros) identificado através do endereço IP. Assim, basta uma simples consulta a bases de dados e a serviços de localização, como o Whois[1], para descobrir onde o usuário está e de qual local ele está fazendo o acesso. Outro modo de identificar usuários é através de Cookies, pequenos pedaços de texto (arquivos) que os sites fazem o navegador do usuário salvar com o objetivo de identificá-lo em um futuro acesso. 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 o endereço eletrônico (caso do spam) do usuário ou seus hábitos de navegação e de consumo a sites de anúncios. Assim, a capacidade de rastrear um usuário se tornou uma atividade altamente lucrativa para empresas especializadas em anúncios.

Empresas de publicidade on-line atuam em parceria com vários sites para coletar dados dos usuários durante a navegação e assim constroem um perfil detalhado dos interesses e atividades desses usuários. Mesmo que alguns usuários não vejam problemas nisso (que mal existe em receber anúncios on-line relevantes?), a posse de dados de usuários, coletados sem consentimento, por empresas de publicidade on-line ou mesmo por qualquer empresa ou agência governamental que queria comprá-los, traz consequências potencialmente desastrosas para a privacidade das pessoas e, em casos mais graves, pode envolvê-las em roubos, fraudes e ataques maliciosos.

Várias tentativas de solucionar o problema foram propostas. Em 1997, um padrão para o uso de Cookies foi especificado na RFC 2109 [Kristol and Montulli 1997], no qual proibia o uso de Cookies de terceiros. Embora, na época, nenhum dos dois navegadores tradicionais, Netscape *Navigator* e Internet Explorer, tenham adotado a especificação, o

[1]Whois é um protocolo voltado para consulta de informações sobre domínios

padrão vingou e hoje, por exemplo, um recente estudo sobre Cookies, feito pela comScore [Weinberger 2011], mostra que aproximadamente um em cada três usuários apagam os Cookies dentro de até um mês após visitarem um site.

Em 2005, os desenvolvedores de navegadores começaram a adicionar um modo de navegação privada em seus produtos. Tal solução dava aos usuários a opção de visitar sites sem deixar Cookies de longo prazo. Desenvolvedores independentes também começaram a produzir extensões para preservar a privacidade dos usuários. AdBlock Plus[2], uma extensão para o navegador Firefox que rejeita anúncios e Cookies de terceiros é um bom exemplo desse tipo de solução. Hoje, ferramentas mais modernas como Ghostery[3] e Lightbeam[4] revelam o número de rastreadores existentes em cada site e mostram como esses rastreadores colaboram com locais (sites) aparentemente não relacionados.

Entretanto, os interessados em continuar nesse "ramo de atividade" não ficaram parados. Aproveitando-se do surgimento e proliferação dos dispositivos móveis, com possibilidades de comunicação e uso cada vez maiores, eles mudaram a forma de rastreamento. 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 de usuários através de seu próprio conjunto de hardware/software. Esse conceito é conhecido como *Device Fingerprinting* e 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 et al. 2014], *device 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.

As técnica de *device fingerprinting* estão se tornando comuns e, ainda assim, muitos usuários sabem pouco ou quase nada sobre o assunto. Mesmo quando estão cientes de que estão sendo monitorados, por exemplo, como uma medida de proteção contra a fraude, eles, em essência, simplesmente confiam que as informações coletadas não serão utilizadas para outros fins.

Assim, o objetivo deste Capítulo é desmarcar o mundo por trás do *device fingerprinting*. A ideia é apresentar conceitos, tecnologias e métodos de desenvolvimento, bem como o alcance e o consequente perigo representado por tais técnicas. Além de possibilitar a compreensão dos principais problemas e desafios a serem superados no desenvolvimento e captura de informações, especialmente via Web, através de técnicas de *device fingerprinting*, este Capítulo identifica oportunidades de estudos na área, bem como aponta alguns trabalhos futuros, visto que o emprego de tais técnicas deve crescer ainda mais nos próximos anos [Tanner 2013].

Para alcançar os objetivos propostos, o restante deste Capítulo está organizado da seguinte forma. A Seção 2.2 caracteriza *device fingerprinting*, apresentando conceitos, definições, classificaçoes e uma breve discussão sobre os problemas de privacidade causados. A Seção 2.3 trata do ambiente porta de entrada para um *device fingerprinting*, o navegador. Questões como a guerra entre os fabricantes, a arquitetura e como é possível identificar um navegador são relatadas. Na Seção 2.4 são discutidas as principais tec-

[2]https://adblockplus.org

[3]https://www.ghostery.com/

[4]https://addons.mozilla.org/pt-br/firefox/addon/lightbeam/

nologias empregadas (ou melhor dizendo, alvo) em um *device fingerprinting*. HTML5, JavaScript, Flash, Canvas, CSS, WebGL e Silverlight são avaliadas em termos de funcionalidades e potencial para obtenção de informações sem consentimento do usuário. Exemplos e trechos de código também são apresentados.

Embora o foco deste Capítulo não seja ensinar o leitor a realizar atividades de *fingerprinting*, a Seção 2.5 apresenta um roteiro para se construir um ferramenta simples de *device fingerprinting*. Como prova de conceito, são expostos alguns resultados obtidos de um site público de *device fingerprinting*. A Seção 2.6 discute as soluções existentes que realizam *fingerprinting*, onde trabalhos acadêmicos e soluções comerciais são apresentadas. Já a Seção 2.7 faz o caminho inverso. Apresenta as contramedidas contra *fingerprinting*. Esta seção tem o intuito de apresentar ferramentas, aplicações e soluções disponíveis (implementadas) na contenção de *device fingerprinting*. Por fim, a seção 2.8 apresenta os comentários finais sobre o tema e aponta algumas questões em aberto.

2.2. Device 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[5] surgiram nessa época e são bastante utilizadas até hoje.

Num âmbito mais voltado a Web e foco deste Capítulo, o termo *fingerprinting* veio a tona em 2009, quando Mayer [Mayer 2009] observou que as características de um navegador e seus plug-ins podiam ser identificadas e os usuários rastreados, e quando, em 2010, o trabalho de Peter Eckersley [Eckersley 2010] mostrou como informações (atributos) fornecidas pelo navegador dos usuários eram suficientes para identificar a grande maioria das máquinas que navegam na Internet. Eckersley desenvolveu um algoritmo para investigar o grau em que os navegadores modernos estão sujeitos as técnicas de *device fingerprinting*. Dos mais de 470.000 usuários que participaram de seu projeto público Panopticlick[6], 84% tiveram seus navegadores identificados (“fingerprintados”).

Formalmente, o termo *fingerprint* foi definido na RFC 6973 [Cooper et al. 2013] como “um conjunto de elementos de informação que define 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”. Este Capítulo partilha da visão que *fingerprinting* é parte de um conjunto amplo de tecnologias e técnicas, também conhecidas como *Device Intelligence*, *Machine Fingerprinting*, *Browser Fingerprinting*, *Web Fingerprinting* ou *Device Fingerprinting*, usadas para identificar (ou reidentificar) um usuário ou um dispositivo, através de um conjunto de configurações, atributos (tamanho da tela do dispositivo, versões de software instalado, entre muitos outros) e outras características observáveis durante comunicações. Neste Capítulo, os termos *device fingerprinting* e *fingerprinting* serão usados para representar essas técnicas de

[5]http://nmap.org

[6]http://panopticlick.eff.org

identificação com foco na Web.

Dois aspectos interessantes e relevantes podem ser observados sobre *device fingerprinting*. O primeiro é que embora tenham sido popularizadas no ambiente Web, tais técnicas estão presentes no mundo móvel. Trabalhos como [Giura et al. 2014] e [Goodin 2013] 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, *device fingerprinting* tem um grande potencial de ameaça a privacidade dos usuários na Web. As questões relacionadas aos problemas de privacidade serão discutidas na Seção 2.2.2. É importante também ressaltar que os trabalhos apontam o uso dessas técnicas para engendrar ataques (persistentes e direcionados) [Forshaw 2011, Contextis 2011].

2.2.1. Classificação

De acordo com o W3C [W3C 2014], as técnicas de *device fingerprinting* podem ser classificadas em três tipos:

1. **Passivo**: Também chamado de *browser fingerprinting*, é aquele baseado nas características observáveis no conteúdo de solicitações Web, sem a utilização de qualquer código em execução no lado do cliente. Eventualmente, esse tipo de *fingerprinting* inclui Cookies, bem como o conjunto de cabeçalhos de solicitação HTTP, endereço IP e outras informações do nível de rede.

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

3. **Cookie-like**: Nessa categoria, usuários, *user-agents*[7] e dispositivos também podem ser reidentificados por um site que primeiro configura e depois recupera o estado armazenado pelo *user-agent* do navegador ou dispositivo. Permite a reidentificação de um usuário ou inferências sobre um usuário da mesma forma que os Cookies permitem o gerenciamento de estado para o protocolo HTTP (RFC6265 [Barth 2011]). Também podem contornar as tentativas do usuário em limitar ou apagar os Cookies armazenados pelo *user-agent*, como demonstrado em [Kamkar 2010].

2.2.2. Privacidade

Embora grande parte do desenvolvimento das técnicas de *device fingerprinting* esteja relacionada a identificação de usuários como medida de segurança e mecanismo anti-fraude (empresas como BlueCava[8], Iovation[9] e ThreatMetrix[10] 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 [W3C 2014]:

[7] *user-agent* é um componente do cabeçalho do protocolo HTTP.
[8] http://www.bluecava.com
[9] http://www.iovation.com
[10] http://www.threatmetrix.com

1. **Identificação do usuário**: Até por questões de padronização, a maioria dos usuários preferem ficar anônimos quando navegam na Internet. Segurança física e pessoal, discriminação, sigilo de dados, entre outros, são motivos para os usuários ficarem no anonimato. Mas, 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. [W3C 2014] afirma que tais inferências permitem oferecer e mostrar ao usuário produtos em determinada faixa de 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.

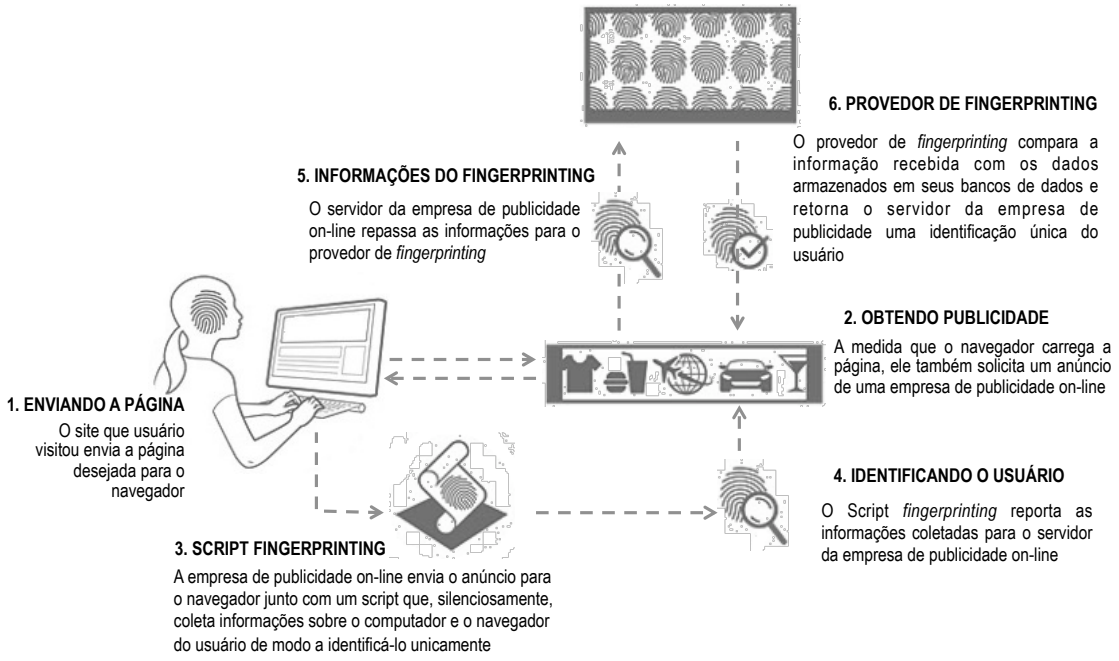

Figura 2.1. Exemplo do processo de *fingerprinting*. Fonte: [Nikiforakis et al. 2014]

2.3. Navegador: tudo começa com ele

Nos dias de hoje, é impossível negar e contestar a presença dos navegadores Web (normalmente chamados de *Browsers* e a partir deste ponto tratados apenas como navegadores) na rotina dos usuários de qualquer sistema 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 [Mulazzani et al. 2013]. A Microsoft tem o Internet Explorer, a Apple criou o Safari, o 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 [StatCounter 2014] aponta o navegador Chrome como o mais utilizado entre os internautas no período de Janeiro a Agosto de 2014, englobando 46,26% da preferência dos usuários. Em seguida vem o Internet Explorer (IE) com 20,31%, o Firefox com 17,5%, o Safari com 10,81% e o Opera com 1,47%.

A Figura 2.2 apresenta o gráfico de uso dos cinco (5) navegadores mais populares nos últimos cinco (5) anos.

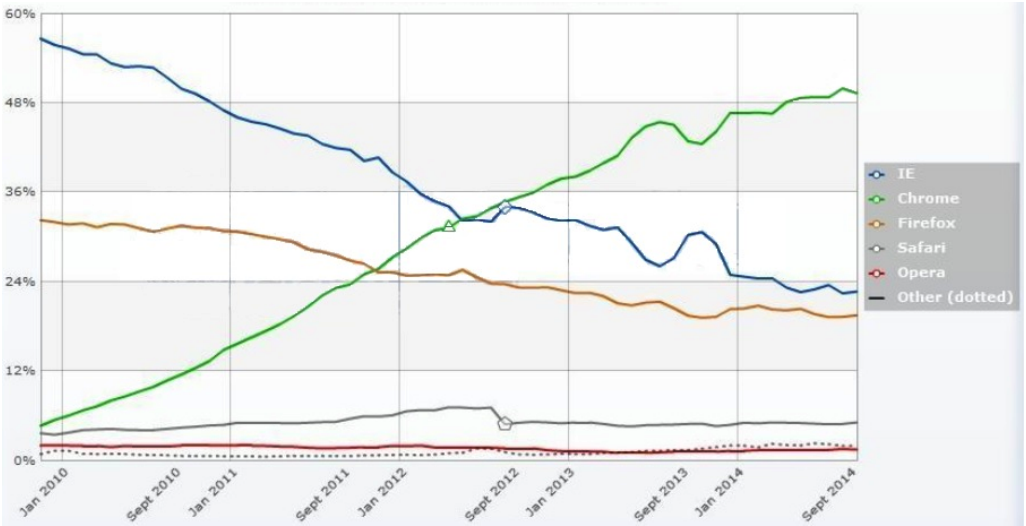

Figura 2.2. Crescimento dos 5 maiores navegadores nos últimos 5 anos. Fonte: [StatCounter 2014]

No que diz respeito ao *device fingerprinting*, o problema com os navegadores é a falta de padrão. Toda a interação e a forma como o navegador interpreta e exibe os recursos solicitados pelo usuário segue padrões definidos e mantidos pelo W3C (*World Wide*

Web Consortium)[11]. 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 trás implicações relevantes na segurança e privacidade dos usuários e informações. É exatamente neste ponto que entram ou se encaixam as técnicas de *device fingerprinting*.

2.3.1. Arquitetura do Navegador

A arquitetura de um navegador une diversas funcionalidades de forma a tornar capaz de apresentar as páginas Web, mas, para que isso ocorra, alguns itens são imprescindíveis. Usando a Figura 2.3 como modelo, os componentes de um navegador são [CCEE 2012]: Interface de Usuário, Motor de Navegação, Motor de Renderização, Networking, Motor de Interpretação de JavaScript, UI Backend e Data Storage.

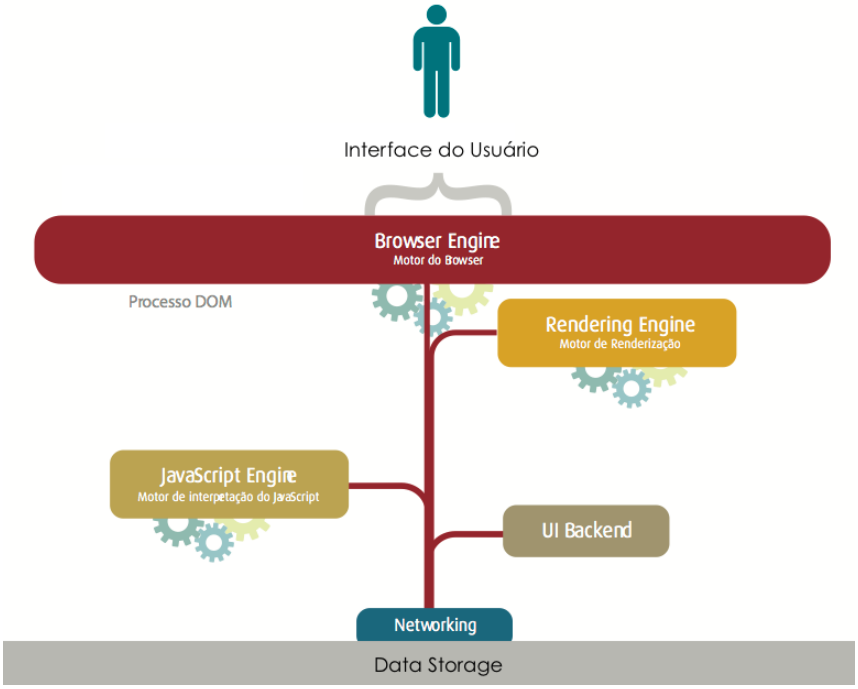

Figura 2.3. Arquitetura simplificada de um navegador. Fonte [CCEE 2012]

A **Interface de Usuário** é o espaço de interação entre os usuários e o navegador. Barra de endereços, botões de avanço e retorno, página inicial, atualização, favoritos, entre outros, são os elementos da interface do usuário (UI). Em outras palavras, tudo o que o navegador exibe faz parte desse componente, exceto pela janela onde o usuário vê a página requisitada.

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*

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

Engine). O papel do motor de navegação é consultar e manipular o motor de renderização, de acordo com as requisições que ocorrem entre a aplicação e a interface de usuário.

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, XML e imagens. Pode também exibir outros tipos de dados através de plug-ins ou extensões, isto é, para exibir documentos PDF é preciso um plug-in 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, um fork do WebKit. Por fim, o motor de renderização está interligado com execuções do interpretador de JavaScript em processos que ocorrem em tempo de execução.

Networking é a parte do código do navegador responsável por gerenciar as chamadas de rede, como, por exemplo, o envio de solicitações HTTP para o servidor.

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

UI Backend é a parte do código usada para desenhar os elementos de design básicos do navegador como caixas de combinação e janelas. Este *backend* expõe uma interface genérica que não é plataforma específica. Ela usa métodos de interface de usuário do sistema operacional.

O **Data Storage** é um componente persistente onde o navegador pode salvar todos os tipos de dados locais, como arquivos diversos, cache, Cookies, entre outros. Os navegadores também suportam mecanismos de armazenamento tais como banco de dados indexados, WebSQL e sistemas de arquivo.

2.3.2. Como identificar um 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 hospeda), 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 [Fielding et al. 1999], *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. Uma típica resposta ao *user-agent* de um navegador é a seguinte:

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.78.2 Version/7.0.6 Safari/537.78.2

A Tabela 2.1 explica os dados obtidos do *user-agent* anterior.

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 plug-in *UAPick*[12]. Para o Firefox, o U*ser Agent Switcher*[13] é o plug-in mais usado. A mesma extensão é usada no Chrome[14]. O Safari possui o *User Agent Browser*[15]. 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*

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 (*Document Object Model*) 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 [W3C 2014] diz que qualquer informação sobre o navegador é feita através do atributo *navigator* da interface *Window*, que deve retornar uma instância da interface *Navigator*.

A Listagem 2.1 apresenta um simples exemplo, em JavaScript, de como obter dados para *device fingerprinting* via objeto *Navigator*. No exemplo, são obtidas informações do *user-agent*, da linguagem do navegador, da plataforma e a classe da CPU. Esta última propriedade só funciona para navegadores IE, o que é uma característica que permite a fácil identificação do navegador.

Listagem 2.1. JavaScript para obter os dados do *user-agent*

```
1  function fingerprint_useragent() {
2     "use strict";
3     var strSep, strTmp, strUserAgent, strOut;
4
5     strSep = "|";
6     strTmp = null;
7     strUserAgent = null;
8     strOut = null;
```

[12]http://www.enhanceie.com/ietoys/uapick.asp

[13]https://addons.mozilla.org/pt-BR/firefox/addon/user-agent-switcher/

[14]https://chrome.google.com/webstore/detail/user-agent-switcher-for-c/djflhoibgkdhkhhcedjiklpkjnoahfmg

[15]https://itunes.apple.com/br/app/user-agent-browser-simply/id414229165?mt=8

```
9
10      /* navigator.userAgent is supported by all major browsers */
11      strUserAgent = navigator.userAgent.toLowerCase();
12      /* navigator.platform is supported by all major browsers */
13      strTmp = strUserAgent + strSep + navigator.platform;
14      /* navigator.cpuClass only supported in IE */
15      if (navigator.cpuClass) {
16         strTmp += strSep + navigator.cpuClass;
17      }
18      /* navigator.browserLanguage only supported in IE, Safari and Chrome */
19      if (navigator.browserLanguage) {
20         strTmp += strSep + navigator.browserLanguage;
21      } else {
22         strTmp += strSep + navigator.language;
23      }
24      strOut = strTmp;
25      return strOut;
26   }
```

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
user-agent	Retorna o cabeçalho do *user-agent* enviado pelo navegador para o servidor

2.4. Tecnologias empregadas

Os navegadores se tornaram as plataformas mais sofisticadas para execução de aplicações, assumindo mais funcionalidades do que as tradicionalmente fornecidas pelo sistema operacional [Mowery and Shacham 2012]. Grande parte deste aumento de sofisticação foi impulsionado por tecnologias e conjuntos de especificações que fornecem a capacidade de desenhar dinamicamente em área da tela (<Canvas>), gráficos tridimensionais (WebGL), armazenamento de dados estruturados do lado do cliente, serviços de geolocalização, capacidade de manipular o histórico e o cache do navegador, reprodução de áudio e vídeo, e muito mais.

Esta seção irá explicitar as principais tecnologias utilizadas nos navegadores, dentre as quais destacam-se: JavaScript, Flash Player, Canvas, WebGL, CSS e Silverlight, todas alvo de *device fingerprinting*. Contudo, antes de iniciar a apresentação de cada uma delas, é preciso falar sobre o padrão HTML5.

2.4.1. HTML5

A HTML5 é a quinta versão do padrão HTML (*Hypertext Markup Language*), que desde de dezembro 2012 recebeu o status de "Candidata a Recomendação" do W3C. Assim como suas sucessoras, 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 [Pemberton 2002] e o HTML DOM Nível 2 [Hors et al. 2003]. 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. Embora as especificações da HTML5 ainda não tenham sido finalizadas e estejam sujeitas as mudanças, os principais navegadores, especialmente o Mozilla, começaram a implementar partes deste padrão. Pelas mesmas razões já citadas, HTML5 também é um candidato em potencial para aplicações móveis multiplataforma. Muitos recursos da HTML5 foram construídos com o requisito de serem capazes de executar em dispositivos de baixa potência, como smartphones e tablets.

A Figura 2.4 exemplifica as tecnologias e mudanças no HTML5.

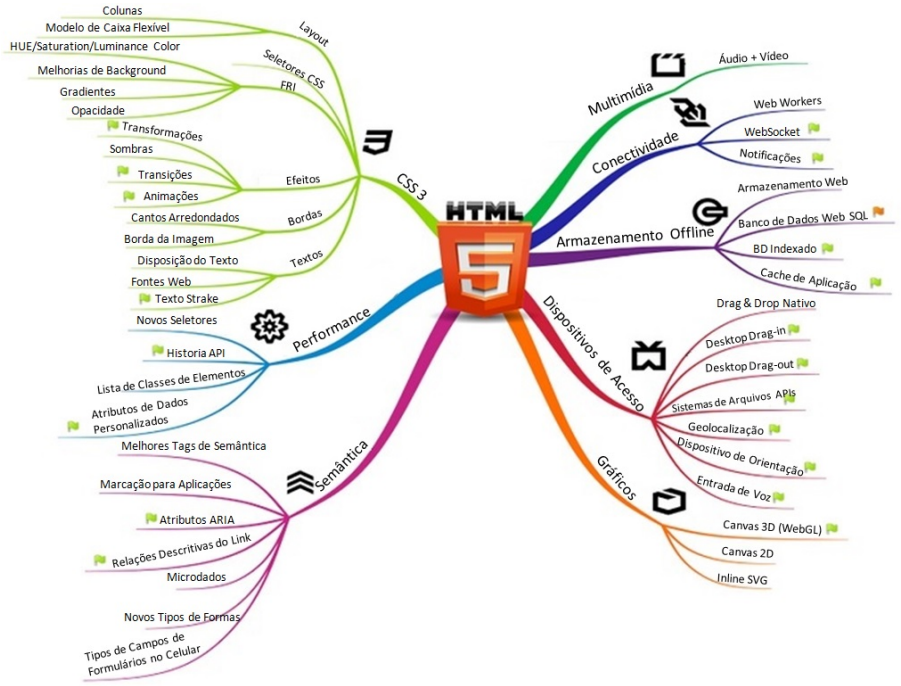

Figura 2.4. Funcionalidades e Características do HTML5

Contudo, as funções melhoradas da HTML5, visando fornecer aos usuários a experiência de um aplicativo nativo em um tempo significativamente menor e a um bom custo, também trouxeram problemas como *fingerprinting*. A título de exemplo, a Listagem 2.2 apresenta dois trechos de código, em JavaScript, que utilizam uma biblioteca capaz de detectar o suporte a muitas das características do HMTL5 e CSS3, a Modernizr [Modernizr 2014].

O primeiro trecho verifica se o navegador tem suporte para trabalhar off-line, ou seja, se aplicações Web podem operar off-line, sem conexão com a Internet. Na prática, a capacidade de operar off-line começa como aplicações Web on-line. A primeira vez que

o navegador visita um site habilitado para off-line, o servidor Web informa ao navegador quais arquivos que ele precisa para trabalhar off-line. Uma vez que o navegador baixou todos os arquivos necessários, o site pode ser acessado (revisado) mesmo sem conexão com à Internet.

O segundo trecho verifica se o navegador suporta a API de geolocalização. Se sim, haverá uma propriedade de geolocalização no objeto *Navigator*.

Listagem 2.2. Trechos de exemplos para checar o suporte a propriedades do HMTL5

```
if (Modernizr.applicationcache) {
  // window.applicationCache is available!
} else {
  // no native support for off-line :(
  // try a fallback or another third-party solution
}
...
if (Modernizr.geolocation) {
  // let's find out where you are!
} else {
  // no native geolocation support available :(
  // try geoPosition.js or another third-party solution
}
```

2.4.2. 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 [Crockford 2008]. 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.

O JavaScript pode ser ativado ou desativado nos navegadores. Assim, uma vez desativado poderá prevenir técnicas de *fingerprinting*. Entretanto, ao se desativar o JavaScript, partes do conteúdo de sites, como vídeos e gráficos interativos, deixam de ser exibidas.

Mas como informações do usuário podem ser obtidas com o Javascript habilitado? De modo geral, através de um objeto DOM pode-se revelar informações importantes a respeito do navegador, tais como o *user-agent*, a arquitetura, o idioma do sistema operacional, o tempo do sistema, a resolução de tela, entre outros.

As Listagens 2.3 e 2.4 apresentam duas simples funções de exemplo do uso de JavaScript para *device fingerprinting*. A primeira descobre se a conexão está sendo feita via um Proxy Web, uma tentativa de garantir anonimato na Internet. A segunda lista os plug-ins instalados no navegador.

Listagem 2.3. JavaScript para detecção de Proxy Web

```
var our_proto = "https";
var our_host = "zorrovpn" + "." + "com"; // mask address
var our_request = "show-js-ip";

// Detect web-proxy version
var webproxy = "No";

if (window["_proxy_jslib_SCRIPT_URL"]) {
```



```
  webproxy = "CGIProxy (" + window["_proxy_jslib_SCRIPT_URL"] + ")";
} else if (window["REAL_PROXY_HOST"]) {
  webproxy = "Cohula (" + window["REAL_PROXY_HOST"] + ")";
} else if (typeof ginf != 'undefined') {
  webproxy = "Glype (" + ginf.url + ")";
} else if (window.location.hostname != our_host) {
  webproxy = "Unknown (" + window.location.hostname + ")";
}

// Trick for CGIProxy
window["_proxy_jslib_THIS_HOST"] = our_host;
window["_proxy_jslib_SCRIPT_NAME"] = "/" + our_request + "?#";
window["_proxy_jslib_SCRIPT_URL"]
  = our_proto + "://" + our_host + "/" + window["_proxy_jslib_SCRIPT_NAME"];

document.write("<b>Detected IP address:</b> ");
document.write('<script sr');
document.write('c="' + our_proto + ":" + '//' + our_host + '/' + our_request);
document.write('"><\/script>');
document.write("<br><b>Web-proxy:</b> " + webproxy);
```

Listagem 2.4. JavaScript para enumerar e listar os plug-ins instalados

```

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

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

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

        if (navigator.javaEnabled()){
            found['java'] = '';
        }
    } else {
        var plug-ins = navigator.plug-ins;
        var reg = /Flash|PDF|Java|Silverlight|QuickTime/;
        for (var i = 0; i < plug-ins.length; i++) {
            var reg_val = reg.exec(plug-ins[i].description);
            if (reg_val){
                var plug-in = 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 = plug-ins[i].version ||
                    (plug-ins[i].name + ' ' + plug-ins[i].description);
                var version_reg_val = version_reg.exec(version);
                if (!found[plug-in]) {
                    found[plug-in] = (version_reg_val && version_reg_val[0]) || '';
                }
            }
        }
    }

    return found;

```



```
48      /* Return version number if plug-in installed
49       * Return true if plug-in is installed but no version number found
50       * Return false if plug-in not found */
51      function msieDetect(name){
52          try {
53              var active_x_obj = new ActiveXObject(name);
54              try {
55                  return active_x_obj.GetVariable('$version');
56              } catch(e) {
57                  try {
58                      return active_x_obj.GetVersions();
59                  } catch (e) {
60                      try {
61                          var version;
62                          for (var i = 1; i < 9; i++) {
63                              if (active_x_obj.isVersionSupported(i + '.0')){
64                                  version = i;
65                              }
66                          }
67                          return version || true;
68                      } catch (e) {
69                          return true;
70                      }
71                  }
72              }
73          } catch(e){
74              return false;
75          }
76      }
77  }
```

Dentre os argumentos para empregar JavaScript em *device fingerprinting*, pode-se citar: (i) JavaScript é uma tecnologia bem estabelecida, padronizado como ECMAScript [ECMA International 2011]; (ii) é suportada por todos os principais navegadores; (iii) funciona em dispositivos móveis; (iv) é utilizado por um percentual muito grande de sites; e (v) é ativado por padrão em todos os principais navegadores. Comparada com outras abordagens de identificação, o uso do JavaScript é mais robusto e não pode ser facilmente portada de um navegador para outro.

Recentemente, dois trabalhos utilizaram JavaScript de forma diferente para obter informações sobre o navegador e os usuários. Em [Mowery et al. 2011], 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. [Mulazzani et al. 2013] propôs uma pesquisa voltada a identificar a segurança de um navegador baseado no uso de um mecanismo de *fingerprinting* do JavaScript. Para tanto, os autores utilizaram o Test262[16], 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

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

em 4 dos mais de 10 mil casos testados, enquanto o Internet Explorer 9 falhou em quase 400 casos de testes.

2.4.3. Flash Player

O Adobe Flash Player é um padrão para entrega de 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 plataformas. Dentre as principais características relacionadas ao plug-in Adobe Flash Player, 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 Player é uma prato cheio para *device fingerprinting*. O estudo de Soltani et al. [Soltani et al. 2010] 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 mais populares sites (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 2.5 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 2.5a, o valor 123 é armazenado em Cookies HTTP e Flash. Quando o usuário remove os Cookies HTTP (Figura 2.5b), 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 2.5c).

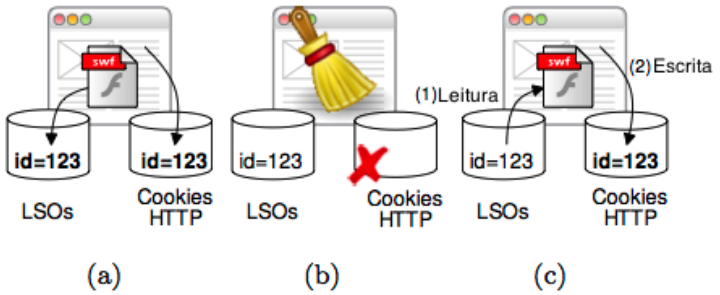

Figura 2.5. Exemplo de *Respawning*: (a) a página Web armazena um Cookie HTTP e um Cookie Flash (LSO), (b) o usuário remove o Cookie HTTP, (c) a página Web "reaparece" com o Cookie HTTP copiando o valor do Cookie Flash. Fonte: [Acar et al. 2014]

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 [McDonald and Cranor 2011]. De modo geral, se um *fingerprinting* conseguir mais ou menos 15 ou 20 bits de informações da identificação, ele pode ter dados

suficientes para identificar o navegador e seu endereço IP.

A Listagem 2.5 representa um ActionScript para determinar o Timezone do computador em Flash.

Listagem 2.5. ActionScript para determinar o Timezone em Flash

```
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      }
```

2.4.4. 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 em [Fulton and Fulton 2011] destacam que Canvas é o equivalente a uma lona utilizada por artistas como superfície de pintura.

A Listagem 2.6 ilustra um código em HTML5 Canvas [Slivinski 2011], cujo resultado é um quadrado vermelho e um texto escrito “Hello World” (Figura 2.6).

Listagem 2.6. Exemplo de Código contendo HTML5 Canvas

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Canvas Test</title>
5      <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
6    </head>
7    <bod>
8      <Canvas id="Canvas" width="200" height="200">
9        Se você estiver vendo isso, o seu navegador não suporta WebGL.
10     </Canvas>
11     <script type="text/javascript">
12       var Canvas = document.getElementById(?Canvas?);
13       var context = Canvas.getContext(?2d?);
14       context.fillStyle = "rgb(255, 0, 0)";
15       context.fillRect(30, 30, 50, 50);
16       context.font = "20px serif";
17       context.fillStyle = "rgb(0, 0, 255)";
18       context.fillText("Hello World", 100, 100);
19     </script>
20   </body>
21 </html>
```

Dentre os principais recursos que o HTML5 Canvas disponibiliza, pode-se destacar:

- **Interatividade**: Canvas pode responder às ações do usuário, através dos eventos de

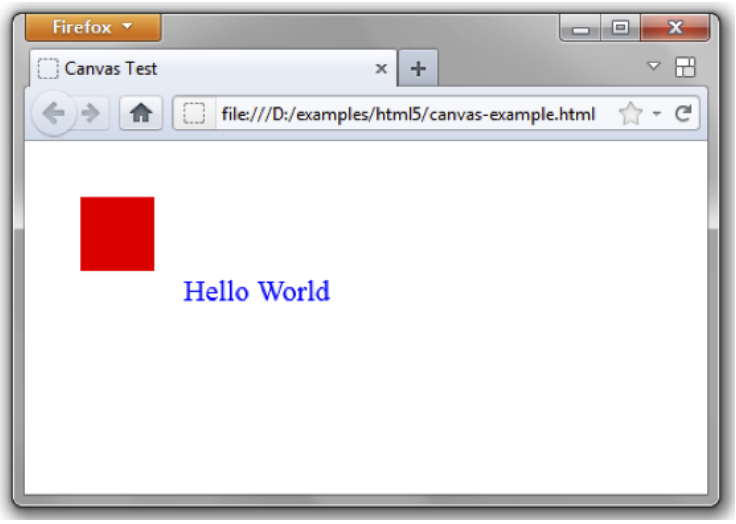

Figura 2.6. Resultado da execução do Código contendo HTML5 Canvas. Fonte: [Slivinski 2011]

teclado, o mouse ou toque;

- **Animação**: Cada objeto desenhado na tela pode ser animado;
- **Flexibilidade**: É possível que os desenvolvedores criem qualquer coisa;
- **Padrão Web**: Canvas é uma tecnologia aberta que faz parte da HTML5;
- **Portabilidade**: Ao contrário do Flash e Silverlight, uma aplicação com Canvas pode ser executada em praticamente qualquer lugar.

Entretanto, HTML5 Canvas faz parte das tecnologias modernas aptas a atuar em *device fingerprinting*. Estudos iniciais mostram que o Canvas pode fornecer uma identificação única rapidamente. O trabalho de Mowery e Shacham [Mowery and Shacham 2012] observou que é possível relacionar o navegador, com maior intimidade, as 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 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 [Kirk 2014] 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 empre-

sas 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 2.7.

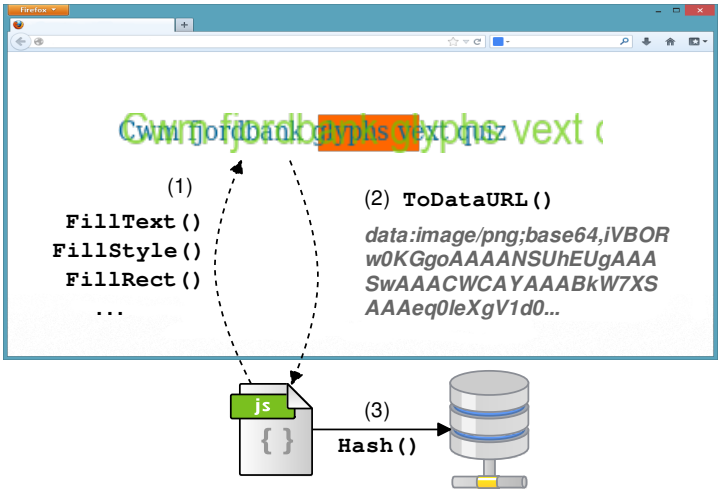

Figura 2.7. Exemplo de *Fingerprinting* utilizando Canvas. Fonte: [Acar et al. 2014]

A Figura 2.7 mostra o fluxo de operações do *fingerprinting* com Canvas. Quando um usuário visita uma página, o script *fingerprinting* primeiro desenha um texto 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 plug-ins, a lista de fontes ou a string *user-agent*.

2.4.5. WebGL

WebGL (*Web Graphics Library*) é uma API multiplataforma que traz a linguagem OpenGL ES 2.0 para a Web, de forma a suportar desenhos 3D dentro do HTML [Group 2014]. Desenvolvida pelo Khronos Group[17], 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 navegadores, mas somente está habilitado no Chrome, Firefox e Opera. O Safari vem com o WebGL desativado por questões de segurança, como explicado a seguir.

Em estudo de 2011, Forshaw [Forshaw 2011] 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 po-

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

dem inutilizar o processamento de toda a máquina. Além disso, os perigos trazidos pela WebGL incluem ataques de negação de serviço [Shankland 2011].

No quesito *device fingerprinting*, Forshaw [Forshaw 2011] 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 [Contextis 2011], 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 2.8 ilustra essa vulnerabilidade e suas consequências.

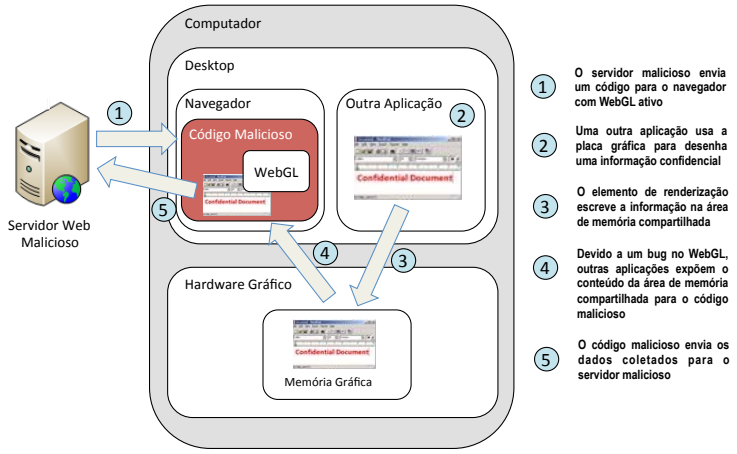

Figura 2.8. Exemplo de um ataque através do WebGL. Fonte: [Contextis 2011]

2.4.6. 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 [Bos et al. 2011]. 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.

Como já mencionado, o principal benefício da CSS é permitir um estilo consistente para ser aplicado em uma série de páginas Web. Já as desvantagens incluem: (i) Problemas de velocidade, já que o download de um arquivo HTML e um arquivo CSS levará mais tempo; (ii) Sintaxe diferente do HTML, e considerada bastante desajeitada e

não amigável para o usuário; e (iii) Templates complicados, mesmo quando associado a sistemas de gerenciamento de conteúdo (CMSs) como Joomla[18] e Drupal[19].

Embora as vantagens do CSS superem suas desvantagens, os projetistas de páginas Web ainda precisam estar cientes dos perigos que podem surgir a partir de seu uso, especialmente no que diz respeito a técnica de *device fingerprinting*. Os autores em [Mulazzani et al. 2013] afirmam que as diferenças nos mecanismos de layout permitem identificar um determinado navegador pelas propriedades CSS que ele suporta. Um caso bem conhecido ocorre quando propriedades CSS ainda estão em status "Candidatas a Recomendação". Assim, os navegadores inserem um prefixo específico antes do fornecedor do motor de layout para indicar que a propriedade é suportada apenas para este navegador. Uma vez que uma propriedade muda para o status "Recomendada", os prefixos são descartados pelo navegador. Por exemplo, no Firefox 3.6 a propriedade *border-radius* teve um prefixo adicionado, resultando em *moz-border-radius*, enquanto que o Chrome 4.0 e o Safari 4.0 usaram *webkit-border-radius*. Quando o Firefox passou a versão 4.0, o Safari para a 5.0 e Chrome também, esse recurso foi uniformemente implementado como *border-radius*.

No site *http://www.caniuse.com* é apresentada uma visão geral de como as propriedades CSS são suportadas nos diferentes navegadores e em seus motores de layout. Assim, uma vez que os navegadores podem diferir em relação a como suportam CSS, seu uso é muito adequado para *fingerprinting* do navegador. Basicamente, existem duas maneiras para testar as propriedades CSS no objeto de estilo de uma página Web. A primeira maneira é simplesmente testar se o navegador suporta uma propriedade específica, usando-a como palavra-chave em um objeto arbitrário. A segunda é olhar para o valor de uma propriedade, uma vez que ela foi definida. Pode-se definir uma propriedade CSS arbitrária em um elemento e consultar o objeto de estilo JavaScript depois. Os valores de retorno indicam se a propriedade de estilo CSS é suportada pelo navegador.

A Listagem 2.7 ilustra um trecho de código JavaScript capaz de obter o histórico da navegação do usuários através do CSS.

Listagem 2.7. Código JavaScript para obter o histórico do navegador via CSS

```
1  var agent = navigator.userAgent.toLowerCase();
2  var is_mozilla = (agent.indexOf("mozilla") != -1);
3
4  // popular websites. Lookup if user has visited any.
5  var websites = [
6    "http://ha.ckers.org",
7    "http://mail.google.com/",
8   ...
9  ];
10
11 /* prevent multiple XSS loads */
12 if (! document.getElementById('xss_flag')) {
13
14   var d = document.createElement('div');
15   d.id = 'xss_flag';
16   document.body.appendChild(d);
17
18   var d = document.createElement('table');
19   d.border = 0;
```

[18]http://www.joomla.org
[19]http://www.drupal.org


```
20    d.cellpadding = 5;
21    d.cellspacing = 10;
22    d.width = '90%';
23    d.align = 'center';
24    d.id = 'data';
25    document.body.appendChild(d);
26
27    document.write('');
28    for (var i = 0; i <> '');
29
30    /* launch steal history */
31
32  if (is_mozilla) {
33    stealHistory();
34  }
35
36  }
37
38  function stealHistory() {
39
40    // loop through websites and check which ones have been visited
41    for (var i = 0; i < websites.length; i++) {
42            var link = document.createElement("a");
43            link.id = "id" + i;
44            link.href = websites[i];
45            link.innerHTML = websites[i];
46            document.body.appendChild(link);
47            var color = document.defaultView.getComputedStyle(link,null).
                  getPropertyValue("color");
48            document.body.removeChild(link);
49  // check for visited
50        if (color == "rgb(0, 0, 255)") {
51            document.write('' + websites[i] + '');
52         } // end visited check
53
54    } // end visited website loop
55
56  } // end stealHistory method
```

2.4.7. 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 [Microsoft 2014], 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 plug-in 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. 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 [Slivinski 2011].

Embora o Silverlight possua várias características importantes e atraentes para os usuários, este plug-in também está vulnerável as técnicas de *device fingerprinting*. Similar as outras tecnologias, é possível obter informações do sistema, tais como: versão do sistema operacional, contagem do processador, fuso horário, fontes instaladas, sistema,

região e idioma do sistema operacional, entre outros.

A Listagem 2.8 ilustra um trecho de código JavaScript para verificar a presença de Silverlight no Navegador. Já a Listagem 2.9 apresenta uma função Silverlight que lista os dispositivos disponíveis no computador. O código usa uma infraestrutura, chamada WIA (*Windows Image Acquisition*), que permite coletar (listar) dispositivos externos como scanners, câmeras de vídeo e câmeras fotográficas conectadas ao computador local. O resultado da função da Listagem 2.9 é apresentado na Figura 2.9.

Listagem 2.8. Verificando a existência de Silverlight no Navegador

```
function fingerprint_silverlight() {
   ...
    try {
        try {
            objControl = new ActiveXObject('AgControl.AgControl');
            if (objControl.IsVersionSupported("5.0")) { strSilverlightVersion = "5.x"
                ; }
            else if (objControl.IsVersionSupported("4.0")) {  strSilverlightVersion =
                "4.x"; }
            else if (objControl.IsVersionSupported("3.0")) {  strSilverlightVersion =
                "3.x"; }
            else if (objControl.IsVersionSupported("2.0")) {  strSilverlightVersion =
                "2.x"; }
            else { strSilverlightVersion = "1.x";  }
            objControl = null;
        } catch (e) {
            objplug-in = navigator.plug-ins["Silverlight Plug-In"];
            if (objplug-in) {
                if (objplug-in.description === "1.0.30226.2") { strSilverlightVersion
                    = "2.x"; }
                else { strSilverlightVersion = parseInt(objplug-in.description[0],
                    10);
                }
            } else \textit{{ strSilverlightVersion = "N/A";  }}
        }
        strOut = strSilverlightVersion;
        return strOut;
    } catch (err) {  return strOnError;  }
}
```

Listagem 2.9. Função Silverlight para listar os dispositivos no computador

```
private void btnIterateWIA_Click(object sender, RoutedEventArgs e)
{
    StringBuilder sb = new StringBuilder();
    sb.AppendLine("List of available external devices and commands:");
    using (dynamic DeviceManager = ComAutomationFactory.CreateObject("WIA.
        DeviceManager"))
    {
        var deviceInfos = DeviceManager.DeviceInfos;
        for (int i = 1; i <= deviceInfos.Count; i++)
        {
            var IDevice = deviceInfos.Item(i).Connect();
            var DeviceID = IDevice.DeviceID;
            var DeviceName = IDevice.Properties("Name").Value;
            var Commands = IDevice.Commands;
            for (int j = 1; j <= Commands.Count; j++)
            {
                var IDeviceCommand = Commands.Item(j);
                var CommandName = IDeviceCommand.Name;
                var CommandDescription = IDeviceCommand.Description;
                sb.AppendLine("    " + DeviceName + " (" + DeviceID + "), " +
                    CommandName + ": " + CommandDescription);
                // Execute with: IDevice.ExecuteCommand(IDeviceCommand.CommandID);
            }
        }
```



```
23          }
24          MessageBox.Show(sb.ToString());
25      }
```

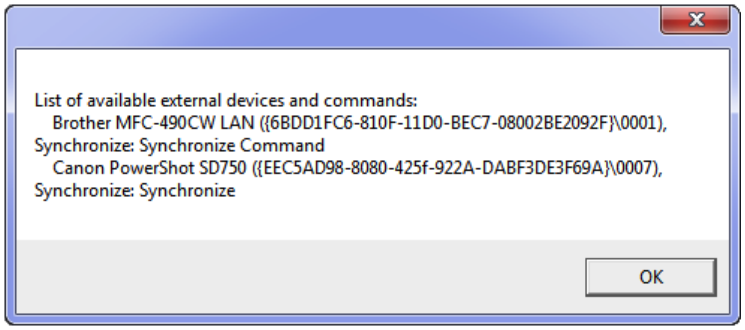

Figura 2.9. Exemplo de lista de dispositivos externos detectados

2.5. Construindo um Device Fingerprinting

Esta seção enumerará os passos necessários para o desenvolvimento de um *device fingerprinting* bem como elucidará os principais dados obtidos durante o processo de *fingerprinting*. Para tanto, a seção será dividida em duas partes. A primeira emprega o trabalho [Nikiforakis et al. 2013] como roteiro para construção de tais mecanismos. Visando melhorar ainda mais o entendimento, a segunda parte apresenta resultados das etapas de um *device fingerprinting*.

2.5.1. Roteiro de Construção

Desde a publicação do trabalho de Eckersley [Eckersley 2010], informações sobre o *user-agent*, o cabeçalho HTTP, a resolução da tela, o timezone, a lista de plug-ins, o sistemas de fontes, entre outras propriedades, tem servido como roteiro para criação de *device fingerprinting*.

Uma vez que todos essas informações (objetos *Navigator* e *Screen* do HTML) já foram discutidas e exemplificadas neste Capítulo, esta seção vai, primeiramente, discutir dois aspectos necessários para melhor se obter (escolher) as informações desejadas. São eles:

- **A escolha de plug-ins populares**: O trabalho de Eckersley [Eckersley 2010] provou que os usuários, tipicamente, possuem vários plug-ins instalados em seus navegadores. Embora a grande maioria deles seja proprietária (o Flash é o melhor exemplo disso), eles possuem ampla adoção, mesmo que as funções que executam agora possam ser realizadas por novos padrões e especificações como a HTML5. No contexto de *device fingerprinting*, o fato é que quanto mais formas de identificar um dispositivo melhor. Por exemplo, um processo de identificação de um usuário Linux, executando o navegador Firefox em uma máquina de 64 bits, responde como "Linux x86-64" quando perguntado (consultado) sobre a plataforma de execução (objeto *Navigator*). Por outro lado, a mesma pergunta feita utilizando um código Flash proporciona uma resposta genérica da versão do kernel (Linux 3.2.0-26-genérico). O mesmo comportamento é visto na chamada que informa a resolução da

tela do usuário. Em implementações Linux do plug-in do Flash (Adobe e Google), quando um usuário utiliza uma configuração com dois monitores (*dual-monitor*), o Flash relata a largura da tela como sendo a soma das duas telas individuais.

- ***Fingerprinting* com foco em vários fabricantes**: Para obter melhores resultados, grande parte dos *fingerprintings* não tentam trabalhar da mesma forma em todos os navegadores. Por exemplo, quando o reconhecimento é focado no Internet Explorer, os códigos buscam extensivamente propriedades específicas como *navigator.securityPolicy* e *navigator.systemLanguage*.

Discutidos estes dois aspectos, a construção de um simples *device fingerprinting* se resume a quatro passos:

1. **Enumeração dos objetos *Navigator* e *Screen***: O objetivo é obter uma lista de todas as propriedades relacionadas a estes objetos disponíveis no navegador. Nesta etapa, os dois aspectos discutidos anteriormente precisam ser levados em consideração. Análises realizadas no trabalho de [Nikiforakis et al. 2013] mostram que a ordem das propriedade durante a enumeração dos objetos *Navigator* e *Screen* é sempre diferente entre as famílias de navegadores, versões de cada navegador e, em alguns casos, entre as implementações da mesma versão em diferentes sistemas operacionais. Além disso, apesar de hoje em dia o padrão HTML ser governado pelo W3C e o JavaScript pelo ECMA, os fabricantes de navegadores ainda adicionam novos recursos que não pertencem a nenhuma norma específica. Em muitos casos, os nomes desses novos recursos começam com um prefixo específico do fornecedor, como *screen.mozBrightness* para o Mozilla Firefox e *navigator.msDoNotTrack* para o Microsoft Internet Explorer. A Figura 2.10 ilustra os possíveis dados que podem ser obtidos.

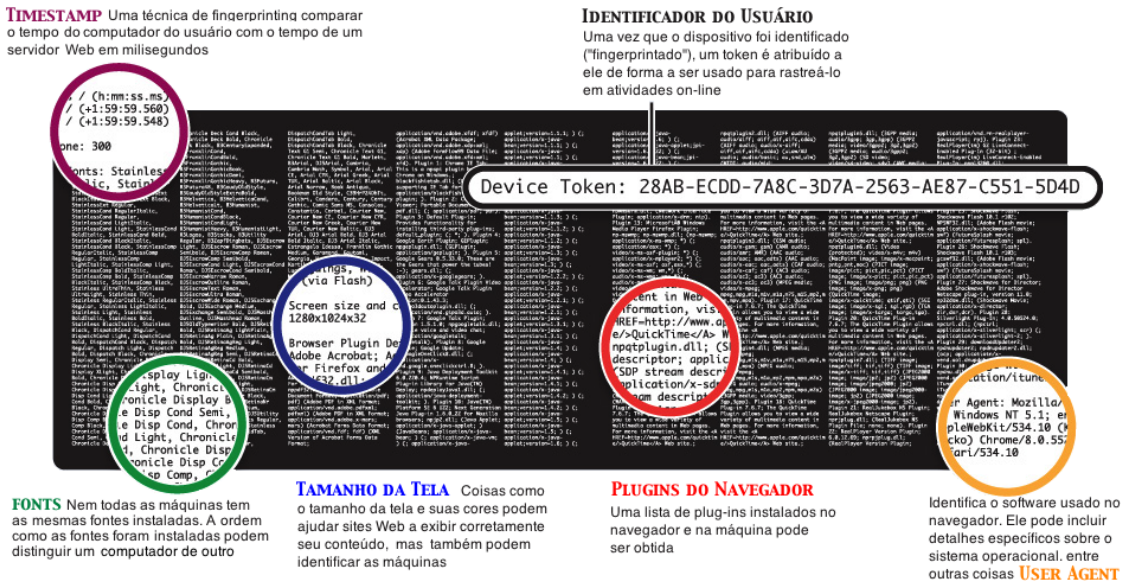

Figura 2.10. Possíveis dados obtidos com um *device fingerprinting*

2. **Enumeração do objeto *Navigator* novamente**: A ideia é verificar se a ordem da enumeração feita no passo anterior não foi modificada, ou seja, se nenhum dos valores recebidos está diferente. Se houverem mudanças, é muito provável que contramedidas *fingerprinting* estejam em uso.

3. **Exclusão e alteração de propriedades dos objetos**: O objetivo é tentar excluir e/ou alterar uma propriedade dos objetos *Navigator* e *Screen*. Isso porque somente o Google Chrome permite que um script exclua uma propriedade a partir do objeto *Navigator*. Já no quesito alteração, somente o Google Chrome e Opera permitem a modificação do valor de uma propriedade do objeto *Navigator*. O mesmo vale para atributos do objeto *Screen*.

4. **Geração do identificador único**: Usando os dados recolhidos pelo script *fingerprinting*, é possível gerar um identificador único para o dispositivo.

2.5.2. Dados coletados durante um device fingerprinting

Além dos diferentes trabalhos já expostos neste Capítulo, esta seção apresenta as principais informações que podem ser facilmente adquiridas, utilizando os conceitos já apresentados. Para isso, o site noc.to foi empregado como modelo.

A primeiras informações "expostas pelo navegador" dizem respeito ao **Sistema**, tais como: (i) Endereço IP e porta; (ii) Sistema operacional; (iii) *user-agent* do navegador; (iv) *user-agent* do sistema operacional; (v) Número de processadores; (vi) Resolução da tela; (vii) Horário do servidor e do cliente.

Em seguida, o *device fingerprinting* do noc.to apresenta informações sobre **Cookies** e o cabeçalho HTTP. Por fim, são relatadas informações JavaScript sobre os objetos *Navigator* e *Screen*, **JavaScript Data** e **JS Display Data**, respectivamente. Vale ressaltar que JavaScript possui muitos campos para ambos objetos e que nem todos os campos existem em todos os navegadores. A Figura 2.11 ilustra todos os tipos dados obtidos.

As próximas informações obtidas são sobre a **Geolocalização do Navegador** e a **Geolocalização do IP** do cliente (Figura 2.12). Para a geolocalização do navegador é utilizada a API JavaScript de gelocalização da HTML5, onde dados como altitude, velocidade e outros valores serão informados se estiverem disponíveis. Já para a geolocalização do IP, utiliza-se informações de geolocalização da MaxMind[20].

Por fim, o noc.to ainda exibe outras informações, incluindo os dados dos plug-ins Silverlight e Flash bem como a lista de plug-ins disponíveis no navegador. Por restrição de espaços, tais dados não serão ilustrados.

2.6. Soluções Existentes

Esta Seção apresenta algumas sites, ferramentas e frameworks que implementam técnicas de *device fingerprinting*. Primeiramente, serão discutidas três soluções acadêmicas, sendo a primeira [Eckersley 2010] considerada a pioneira na área. Depois serão apresentadas soluções comerciais, onde uma discussão, usando o trabalho de [Nikiforakis et al. 2013], revelará mais sobre como e o que essas empresas usam em seus scripts *fingerprinting*.

[20]https://www.maxmind.com/

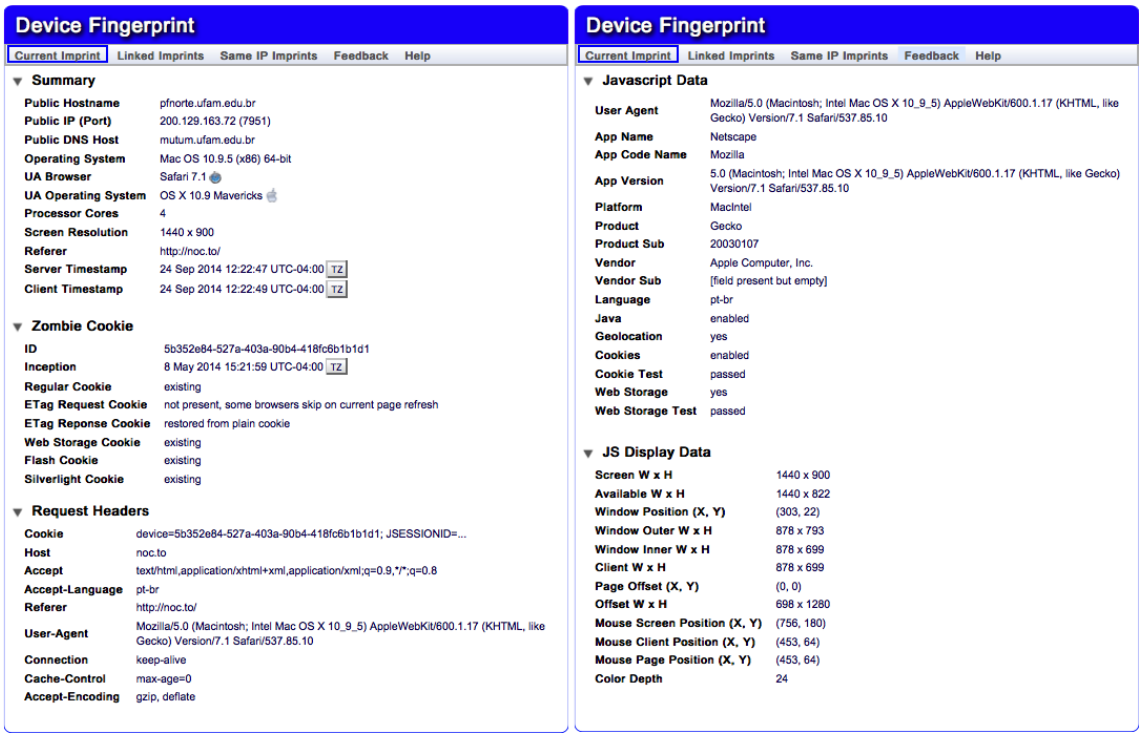

Figura 2.11. Dados referentes ao Sistema, aos Cookies, ao cabeçalho HTTP e JavaScript coletados pelo *device fingerprinting* do noc.to

2.6.1. Pesquisas Acadêmicas

How Unique Is Your Web Browser?

Peter Eckersley [Eckersley 2010], em seu artigo "How Unique Is Your Web Browser", relata sua experiência ao verificar o quanto a configuração de um navegador é única e possível mecanismo de rastreamento de usuários na Internet. Sua pesquisa investigou o grau em que os navegadores modernos estão sujeitos a *device fingerprinting* através da análise das versões e informações de configuração que os navegadores possuem e que podem ser coletadas, com ou sem autorização prévia dos usuários, como, por exemplo, as dimensões de tela, fusos horários e lista de fontes instaladas, entre outras. No final, todas essas informações podem ser combinadas para gerar um identificador único para o dispositivo.

Para realizar a pesquisa, Eckersley desenvolveu um site de teste, denominado **Panopticlick** (https://panopticlick.eff.org) que utiliza um algoritmo, também desenvolvido por ele, cuja finalidade é aplicar mecanismos de *fingerprinting* no navegador do usuário e, consequentemente, gerar um identificador único. Os dados dos navegadores foram recolhidos a partir de visitas feitas ao site de teste. Inicialmente, a pesquisa contou com 470.161 acessos de navegadores operados pelos participantes informados do tema ao visitaram o site.

Como resultado, Eckersley observou que mesmo com os usuários conscientes, foi possível gerar 83,6% de identificações únicas e apenas 5,3% de navegadores/usuários conseguiram se manter no anonimato. Ao verificar se os navegadores possuíam Adobe

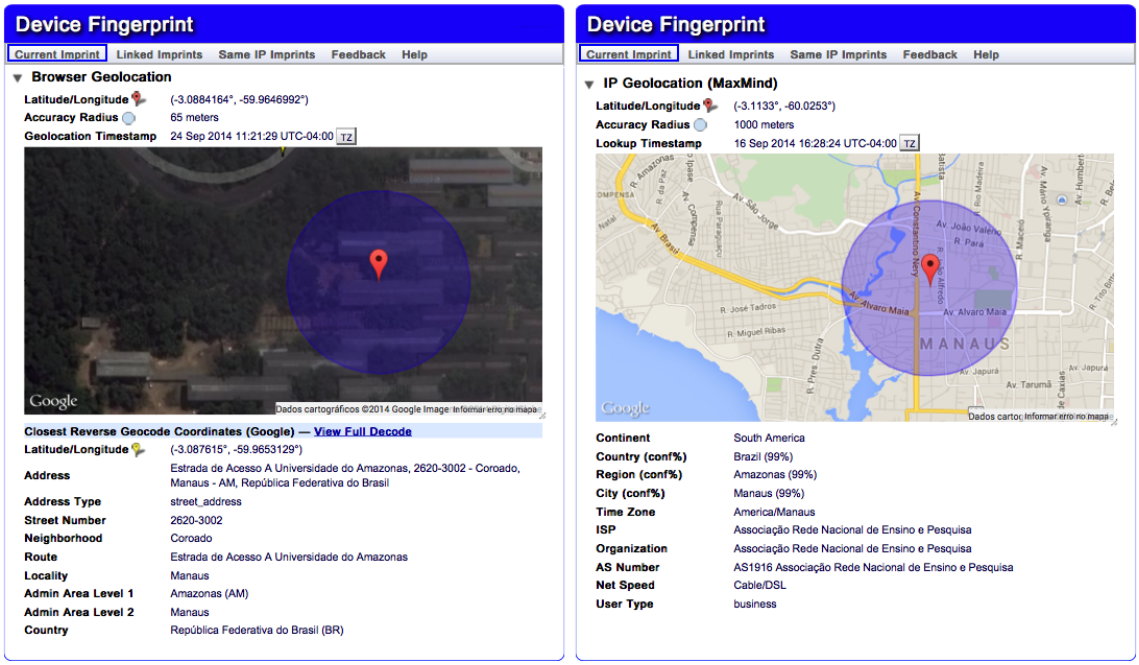

Figura 2.12. Dados referentes a Geolocalização do navegador coletados pelo *device fingerprinting* do noc.to

Flash ou tinham a Máquina Virtual Java habilitada, 94,2% apresentaram instantaneamente identificadores únicos. Apenas 1,0% dos navegadores com Flash ou Java manteve o anonimato. Mas, esses percentuais não são definitivos, pois quando houverem mudanças no navegador como, por exemplo, atualizações de versão, atualização de um plug-in, desabilitação dos Cookies, instalação de uma nova fonte, um aplicativo externo que incluiu fontes ou alterou a resolução da tela, a identificação única feita não será mais válida.

Em particular, Eckersley usou Cookies para reconhecer os navegadores que estavam retornando e assim verificar se suas identificações sofreram mudanças. O preocupante é o fato de um algoritmo, considerado 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%.

O autor também destaca que se o navegador estiver com o JavaScript bloqueado ou se estiver utilizando algum plug-in de bloqueio, o Panopticlick não consegue capturar as informações do navegador. Assim, outros autores, baseados na descoberta de Eckersley, desenvolveram pesquisas e apresentaram outras implementações de *device fingerprinting*.

FPDetective: Dusting the Web for Fingerprinters

A pesquisa de Acar et al. [Acar et al. 2013], descrita no artigo "FPDetective: Dusting the Web for Fingerprinters", relata o desenvolvimento de um framework, denominado FPDetective, que centra-se na detecção de códigos de *fingerprinting*. Ao realizar uma análise em grande escala, com milhões de sites populares da Internet, os autores descobriam que a adoção de técnicas *device fingerprinting* é muito maior do que os estudos anteriores haviam estimado, bem como a lista de códigos de *fingerprinting* conhecidos.

O framework FPDetective está disponível gratuitamente e pode ser obtido a par-

tir do endereço http://homes.esat.kuleuven.be/gacar/fpdetective. A Figura 2.13 ilustra a arquitetura do FPDetctive. O framework foi desenvolvido utilizando Python, C++, JavaScript e MySQL, com o objetivo de ser flexível e poder ser usado para realizar estudos de privacidade Web em larga escala.

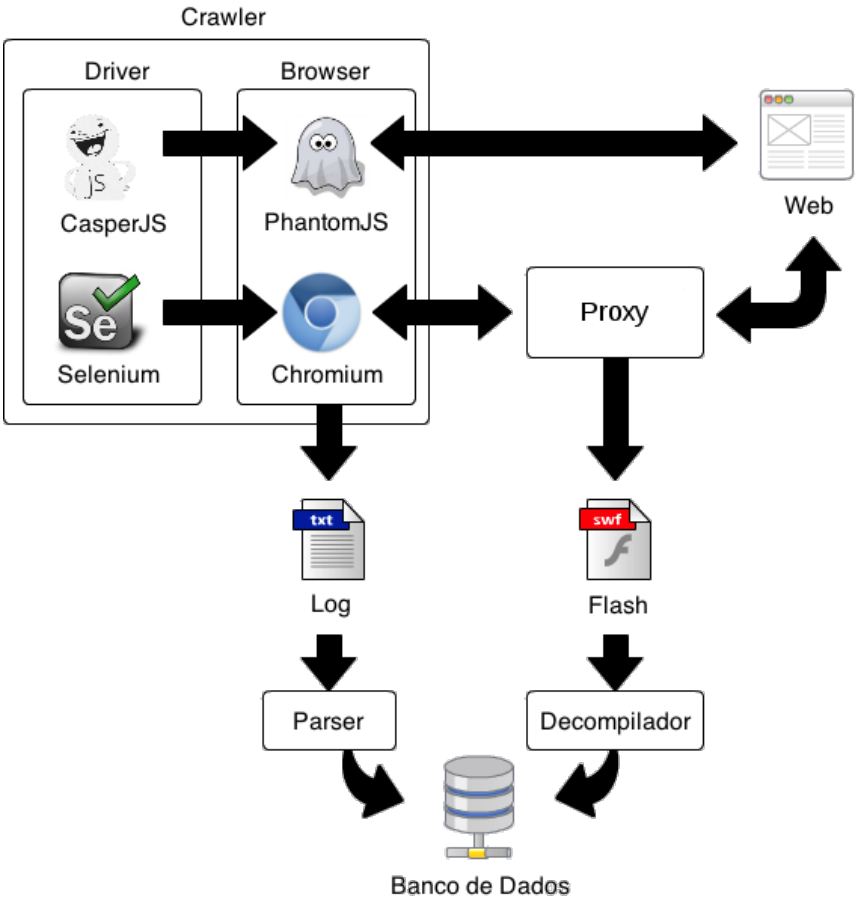

Figura 2.13. Arquitetura do Framework FPDetective

Uma descrição dos componentes do FPDetective é apresentada a seguir:

1. **Crawler**: O Crawler possui dois navegadores, PhantomJS[21] e Chromium[22]. Ambos foram instrumentados, ou seja, tiveram partes do código-fonte do WebKit, o motor de renderização, modificados para se adequar ao trabalho. O PhantomJS foi escolhido para coletar *fingerprinting* baseado em JavaScript, uma vez que faz um uso mínimo de recursos. O Chromium foi usado para investigar *fingerprinting* baseado em Flash, já que o PhantomJS não tem suporte a este plugin. CasperJS[23] e Selenium[24] foram usados para conduzir os navegadores até os sites e navegar através das páginas.

[21] http://phantomjs.org
[22] http://chromium.org
[23] http://casperjs.org/
[24] http://docs.seleniumhq.org/

2. **Parser**: O Parser é usado para extrair dados relevantes, a partir dos dados gerados pelo Crawler, e armazená-los em um banco de dados. Os scripts *fingerprinting* conhecidos e/ou encontrados nas solicitações HTTP também foram processados pelo Parser.

3. **Proxy**: A fim de obter os arquivos Flash para análise estática, todo o tráfego HTTP foi redirecionado para mitmproxy[25], um proxy capaz de interceptar SSL, e a biblioteca libmproxy para analisar e extrair arquivos Flash que fazem *sniffing*.

4. **Decompilador**: A biblioteca JPEXS Free Flash Decompiler[26] foi usada para descompilar arquivos Flash e obter o código-fonte do ActionScript. O Decompilador procura por chamada de funções (*enumerateFonts* e *getFontList*, por exemplo) de *fingerprinting* para obter um vetor binário de ocorrências.

5. **Banco de Dados**: Uma vez que o Crawler pode ser executado em várias máquinas, foi utilizado um banco de dados central para armazenar, combinar e analisar os diferentes resultados com o mínimo de esforço. Os dados armazenados incluem o conjunto de chamadas de funções JavaScript, a lista de pedidos e respostas HTTP e a lista de fontes carregadas ou solicitados. Para os experimentos Flash, também é armazenado um vetor binário que representa a ocorrência de chamadas da API do ActionScript que podem estar relacionados ao *fingerprinting*.

Como resultado da pesquisa, após a verificação realizada pelo FPDetective, foram encontrados 404 sites que estão no Top Rank do site Alexa.com que se utilizam de JavaScript para *fingerprinting*. A Tabela 2.3 apresenta o detalhamento deste resultado.

Tabela 2.3. Resultado do FPDetective para sondagem com JavaScript. Fonte: [Acar et al. 2013]

				Número de sites usando FP baseados em JS		
				1M	100K	
Provedor do Fingerprinting	**Nome do Script**	**No. de Fontes**	**Top Rank**	**Homepage**	**Homepage**	**Página Interna**
BlueCava	BCAC5.js	231/167/62	1.390	250	24	24
Perferencement	tagv22.pkmin.js	153	49.979	51	6	6
CoinBase	application- 773a[...snipped...].js	206	497	28	4	4
MaxMind	device.js	94	498	24	5	5
Inside graph	ig.js	355	98.786	18	1	1
SiteBlackBox	URL não definida	389	1.687	14	10	10
Analytics-engine	fp.js	98	36.161	6	-	-
Myfreecams	o- mfccore.js	71	422	3	1	1
Mindshare Tech.	pomegranate.js	487	109.798	3	-	-
Cdn.net	cc.js	297	501.583	3	-	-
AFK Media	fingerprint.js	503	199.319	2	-	-
Anonymizer	fontdetect.js	80	118.504	1	-	-
				404	**51**	**51**

Já em relação ao Flash, foram encontrados elementos de *fingerprinting* em 145 homepages no Top Rank 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 as capacidades ampliadas para enumeração de fonte, detecção de proxy e compatibilidade com navegadores. A Tabela 2.4 apresenta apenas *fingerprinting* Flash que usam enumeração de fonte (95 deles).

[25]http://mitmproxy.org

[26]http://code.google.com/p/chromium/issues/detail?id=55084

Tabela 2.4. Objetos Flash com enumeração de fontes. Fonte: [Acar et al. 2013]

Provedor do Fingerprinting	No. de Sites	Top Rank	Cookies Flash	Detecção de Proxy	Identificação de HW e SO	Interação com JS
BB Elements	69	903	✓			✓
Piano Media	12	3.532	✓		✓	✓
Bluecava	6	1.390	✓			✓
ThreatMetrix	6	2.793		✓	✓	
Alipay	1	83	✓		✓	✓
meb.gov.tr (Turkish Ministry of Education)	1	2.206	✓		✓	✓

2.6.1.1. SHPF

O SHPF (*Session Hijacking Prevention Framework*) é um framework desenvolvido para combater o sequestro de sessões (*Session Hijacking*) em navegadores Web através de *fingerprinting* [Unger et al. 2013]. Ele oferece um conjunto de múltiplos mecanismos de detecção que podem ser usados independentemente um dos outros. 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 *device 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 2.14 apresenta a arquitetura geral e as funcionalidades básicas do SHPF.

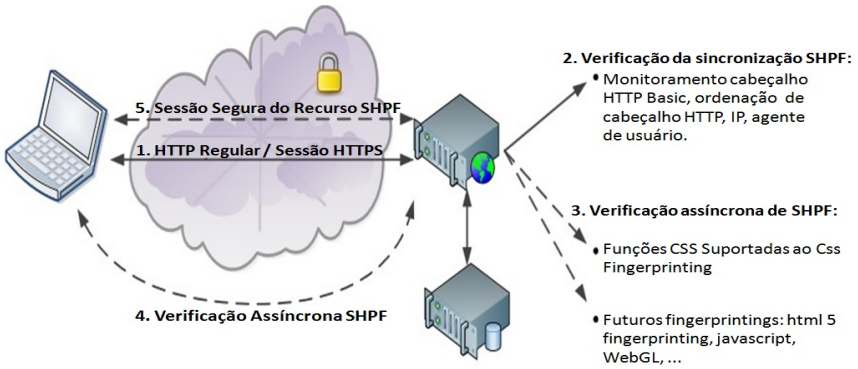

Figura 2.14. Arquitetura e funcionamento do SHPF. Fonte: [Unger et al. 2013]

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 [Unger et al. 2013]. As *features* também são os meios de extender o framework e fornecer classes base para o desenvolvimento rápido de novas *features*. 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.

No que diz respeito ao *fingerprinting*, a *feature* que utiliza um mix de propriedades CSS3 foi capaz de identificar 23 elementos adequados, conforme descrito na Tabela 2.5.

Tabela 2.5. Propriedades CSS e valores identificados pelo SHPF

CSS Status - Recomendado	
Característica	***Valor***
display	nline-block
min-width	35px
position	fixed
display	table-row
opacity	0.5
background	hsla(56, 100%, 50%, 0.3)
CSS Status - Candidato a Recomendação	
Característica	***Valor***
box-sizing	border-box
border-radius	9px
box-shadow	inset 4px 4px 16px 10px #000
column-count	4
CSS Status - Draft	
Característica	***Valor***
transform	rotate(30deg)
font-size	2rem
text-shadow	4px 4px 14px #969696
background	linear-gradient (left, red, blue 30%, green)
transition	background-color 2s linear 0.5s
animation	name 4s linear 1.5s infinite alternate none
resize	both
box-orient	horizontal
transform-style	preserve-3d
font-feature-setting	dlig=1,ss01=1
width	calc(25% -1em)
hyphens	auto
object-fit	contain

2.6.2. Soluções Comerciais

Após o trabalho de Eckersley mostrar a possibilidade de realizar um *fingerprinting* em navegadores Web, a fim de rastrear usuários sem a necessidade de instalar qualquer código ou solução no lado do cliente, ficou claro que esse tipo de atividade já existia e que empresas haviam se especializado nisso. Embora existam várias empresas atuando na

identificação de usuários, todas para fins de autenticação e anti-fraude, três delas (BlueCava, Iovation e ThreatMetrix, todas já mencionadas anteriormente neste Capítulo) são consideradas as grandes, devido sua popularidade e gama de clientes.

A Bluecava (http://www.bluecava.com) é uma empresa que fornece tecnologia de reconhecimento (*fingerprinting*), permitindo às empresas de comércio eletrônico evitar fraudes e melhorar a eficácia de sua publicidade on-line. A tecnologia criada pela empresa identifica e direciona usuários da Internet, sem o uso de Cookies. Além disso, funciona em dispositivos como PCs, smartphones, laptops, consoles de jogos e set-top boxes.

Os serviços e soluções que a BlueCava oferece visam utilizar, principalmente, dados do dispositivo, como o endereço IP, versão do navegador e tipo de dispositivo. Dados de consumo são anonimizados e categorizados em segmentos públicos amplos como "homens 18-34" ou "interesse por esporte", por exemplo. Através de várias implementações, a BlueCava identifica, analisa e sincroniza dezenas de dados de entrada do dispositivo. O resultado é apresentado como um gráfico de associação, que permite aos comerciantes implementarem uma *cross-screen* de segmentação, mensuração e otimização ao seu alcance de audiência e frequência.

A Iovation (http://iovation.com) é uma empresa anti-fraude que têm por objetivo evitar que seus clientes façam negócios com fraudadores. Para isso, é capaz de estabelecer uma identificação única para cada dispositivo e rastrear as atividades, de modo que possam identificar comportamentos fraudulentos. Atualmente, a empresa possui 2 bilhões de informações sobre dispositivos, cobrindo 84% de todas as transações consideradas fraudulentas em sites dos seus clientes. O software responsável por isso é o **Reputation Manager 360**.

Criado em 2004 e em continua evolução desde então, o software é capaz de relatar as empresas se o dispositivo acessando seu site ou aplicativo móvel tem um histórico conhecido de fraude em cartão de crédito, roubo de identidade, entre outros comportamentos abusivos. Também é capaz de informar se o dispositivo em questão está associado com quaisquer outros dispositivos que contenham históricos de fraude.

A última empresa, a ThreatMetrix (http://www.threatmetrix.com/), possui uma plataforma que ajuda a proteger a integridade das transações on-line das organizações (incluindo serviços financeiros e bancários on-line, e-commerce, seguros, redes sociais, governo e grandes empresas). As avançadas tecnologias de identificação de dispositivo, elaboradas pelas empresa, ajudam as organizações a determinar se os visitantes on-line são clientes legítimos ou potenciais criminosos virtuais.

A ThreatMetrix apresenta uma série de aplicativos de defesas, dentre os quais pode-se destacar:

- **TrustDefender**: um plataforma de proteção contra crimes cibernéticos que atua na autenticação baseada em contexto para protege usuários contra fraude de pagamento, registro de conta fraudulenta e ataques de malware.
- **ThreatMetrix SmartID**: uma ferramenta de identificação, que não usa Cookies, que permite criar avançados perfis de identificação exclusivamente baseados nos atributos do dispositivo.

2.6.2.1. Comparação entre as soluções

O trabalho de Nikiforakis et al. [Nikiforakis et al. 2013], no artigo "Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting" relata uma comparação entre as empresas BlueCava, Iovation e ThreatMetrix, apresentadas anteriormente, com o objetivo de investigar de que forma essas empresas estão se utilizando das técnicas de *device fingerprinting*.

Para realizar a análise dos scripts de *fingerprinting* dessas três soluções comerciais, sites foram avaliados para descobrir se utilizam os serviços oferecidos por essas empresas. O software Ghostery, detalhado na próxima Seção, foi utilizado para realizar o rastreamento e obter a lista de domínios que usam os scripts de *fingerprinting*. Assim, para quantificar esses sites populares, 10.000 sites do Alexa.com foram analisadas, com profundidade de até 20 páginas, a procura de inclusões de scripts e *iframe* originários dos domínios das três empresas. No final, descobriu-se que 40 site (0,4% dos 10.000 sites) utilizavam *device fingerprinting* dos três provedores comerciais.

O primeiro resultado obtido pela pesquisa foi uma taxonomia das possíveis funções que podem ser adquiridas através de uma biblioteca de *fingerprinting*. Esta taxonomia abrange todos os recursos descritos por Peter Eckersley [Eckersley 2010] (Panopticlick), bem como as características utilizadas pelas três empresas, conforme mostra a Tabela 2.6.

As categorias propostas nessa taxonomia resultaram da análise em camadas, onde a "camada de aplicação" é o navegador e qualquer informação coletada do navegador. No topo dessa taxonomia estão os scripts que buscam identificar todas as personalizações do navegador. Nos níveis mais baixos estão os scripts que possuem informações específicas do usuário em todo o navegador, o funcionamento sistema e até mesmo o hardware e a rede do usuário da máquina [Nikiforakis et al. 2013].

Tabela 2.6. Comparação entre Panopticlick,BlueCava, Iovation e ThreatMetrix. Fonte: [Nikiforakis et al. 2013]

Categoria	**Panopticlick**	**BlueCava**	**Iovation**	**ThreatMetrix**
Customizações do navegador	Enumeração de plugins (JS); Enumeração de Mime-type (JS); ActiveX + 8 CLSIDs[27] (JS)	Enumeração de plugins (JS); ActiveX + 53 CLSIDs (JS); Google Gears Detection (JS)		Enumeração de plugins (JS); Enumeração de Mime-type (JS); ActiveX + 6 CLSIDs (JS); Fabricante Flash (FLASH)
Configurações do navegador no nível do usuário	Cookies ativos (HTTP); Timezone (JS); Flash ativo (JS)	Linguagem do Sistema/Navegador/Usuário (JS); Timezone (JS), Flash ativo (JS); Do-Not-Track (JS); Política de Segurança MSIE (JS)	Linguagem do navegador (HTTP, JS); Timezone (JS); Flash ativo (JS); Data e hora (JS); Detecção de Proxy (FLASH)	Linguagem do navegador (FLASH); Timezone (JS,FLASH); Flash ativo (JS); Detecção de Proxy (FLASH)
Família e versão do navegador	User-agent (HTTP); ACCEPT-Header (HTTP); Teste de Super Cookie (JS)	User-agent (JS); Constantes matemáticas (JS); Implementação AJAX (JS)	User-agent (HTTP, JS)	User-agent (JS)
Sistema Operacional e Aplicações	User-agent (HTTP), Detecção de fontes (JAVA, FLASH)	User-agent (JS); Detecção de fontes (JS, FLASH); Registro do Windows (SFP)	User-agent (HTTP, JS); Registro do Windows (SFP); Chave dos Produtos MSIE (SFP)	User-agent (JS); Detecção de fonte (FLASH); Versão do sistema operacional e Kernel (FLASH)
Hardware e Rede	Resolução da tela (JS)	Resolução da tela (JS); Enumeração de drivers (SFP); Endereço IP (HTTP); Parâmetros TCP/IP (SFP)	Resolução da tela (JS); Identificação de dispositivos (SFP); Parâmetros TCP/IP (SFP)	Resolução da tela (JS, FLASH)

O resultado da pesquisa revelou que todas as empresas usam Flash, além de JavaScript. Outro fator interessante para identificação do dispositivo através de *fingerprinting* é o endereço IP, visto que a distinção entre um usuário legitimo e outro fraudulento é crucial e pode ser feita localização do IP. Assim, é do interesse do provedor de *fingerprinting* detectar o endereço IP real do usuário ou, pelo menos, descobrir que o usuário está utilizando um servidor proxy. Ao analisar o código do *ActionScript* incorporado ao códigos Flash, de duas das três empresas, encontraram-se evidências de que o código é capaz contornar os proxies definidos pelo usuário no navegador, ou seja, o aplicativo Flash carregado foi capaz de contactar um host remoto diretamente, desprezando qualquer configuração HTTP proxy.

A pesquisa também provou que valores do Registro do Windows (um ambiente rico para *fingerprinting*) podem ser lidos por plugins. *Fingerprinters* submetidos via DLL foram capazes de ler uma infinidade de valores específicos do sistema, tais como identificador do disco rígido, parâmetros de TCP/IP, o nome do computador, identificador de produto, a data de instalação do Windows e os drivers do sistema instalado (entradas marcadas com SFP na Tabela 2.6). Todos estes valores combinados proporcionam uma forte identificação que JavaScript ou Flash jamais poderiam construir.

Por fim, o site mais popular fazendo uso de *fingerprinting* foi o skype.com, enquanto as duas maiores categorias de sites que usam *fingerprinting* são: Pornografia (15%) e Encontros/namoro (12,5%). Para os sites pornográficos, uma explicação razoável é que as *fingerprinting* são utilizadas para detectar credenciais compartilhadas ou roubadas de membros pagantes, enquanto que para sites de namoro é para assegurar que os atacantes não criem múltiplos perfis para fins de engenharia social.

2.7. Contramedidas

Esta seção relatará o que está sendo discutido e utilizado como contramedida para o rastreamento de informações. O uso do Navegador Tor [Tor Project 2014], plug-ins como Firegloves [FireGloves 2014], especificações como Do Not Track (DNT) [Mayer et al. 2011], entre outras, são exemplos do que será apresentado.

2.7.1. Navegador Tor

Tor (*The Onion Router*)[28] [Tor Project 2014] é uma rede de túneis virtuais que permite aos usuários e grupos se comunicarem de forma segura, aumentando assim sua segurança e privacidade na Internet. Ele permite que os desenvolvedores de software criem novas ferramentas de comunicação com recursos internos de privacidade.

Originalmente concebido, implementado e implantado como a terceira geração do projeto *onion routing*, do laboratório de pesquisa Naval dos Estados Unidos, com a finalidade de proteger as comunicações do governo, tornou-se um software livre capaz de permitir que os usuários naveguem na Internet anonimamente, sem que suas atividades e localização possam ser descobertas. Tor redireciona o tráfego Internet através de uma rede mundial de voluntários, formada por mais de cinco mil nós livres, para ocultar a localização do usuário e uso de qualquer pessoa a realização de vigilância de rede ou de

[28] 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.

análise de tráfego. Com isso, torna mais difícil rastrear o tráfego de determinado usuário na Internet, ou seja, visitas a sites, emails, mensagens instantâneas e outras formas de comunicação são protegidas, bem como a liberdade e habilidade para conduzir comunicação confidencial. De modo resumido, permite que organizações e indivíduos partilhem informações através de redes públicas sem comprometer sua privacidade.

Funcionamento

De modo sucinto, Tor criptografa os dados originais, incluindo o endereço IP de destino, várias vezes, e o envia através de um circuito virtual que compreende sucessivas transmissões selecionadas aleatoriamente. Cada transmissão decifra uma camada de criptografia para revelar apenas o próximo nó do circuito. O último nó decodifica a camada mais interna da criptografia e envia os dados originais para o seu destino, sem revelar, ou mesmo saber, o endereço IP de origem. Uma vez que o encaminhamento da comunicação é parcialmente escondido em cada nó co circuito Tor, este método elimina qualquer ponto único em que a comunicação pode ser descoberta através de vigilância de rede, que depende do conhecimento de qual é a origem e o destino.

A Figura 2.15 ilustra o funcionamento do Tor.

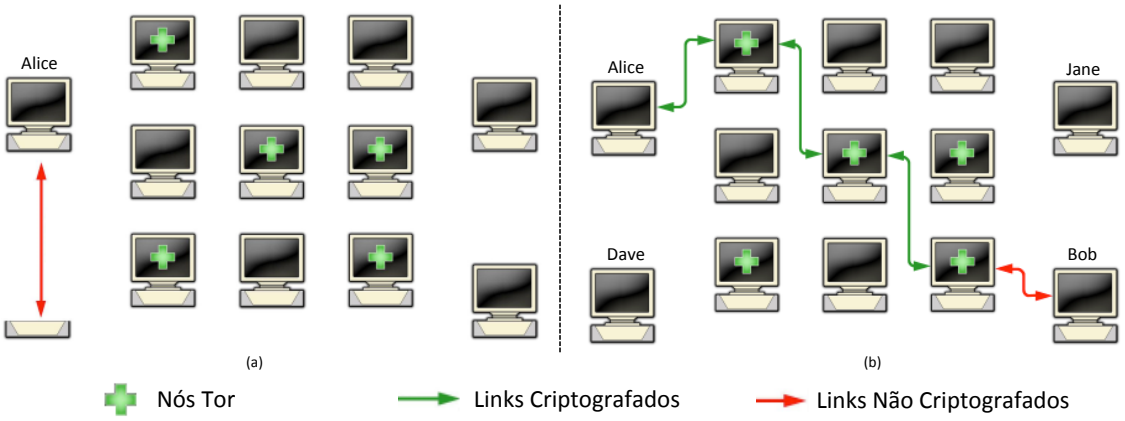

Figura 2.15. Funcionamento do Tor. Fonte: [Tor Project 2014]

Quando Alice quer se comunicar com Bob usando a rede Tor, ela primeiro faz uma conexão não criptografada para um servidor de diretório centralizado que conhece todos os endereços dos nós Tor (a). Após receber a lista de endereços, o cliente Tor na máquina de Alice irá se conectar a um nó aleatório (o nó de entrada) da rede Tor, através de uma conexão criptografada. Por sua vez, o nó de entrada fará uma conexão criptografada a um segundo nó aleatório, que por sua vez faz o mesmo para se conectar a um terceiro nó aleatório. Esse terceiro nó, o nó de saída, se conecta a Bob (b).

É importante ressaltar que um mesmo nó não pode ser usado duas vezes, em uma comunicação. Isso porque se a matriz de nós for utilizada por um longo período de tempo, uma conexão Tor seria vulnerável a análise estatística. Por isso, o software no cliente Tor muda o nó de entrada a cada dez minutos.

Diferente de outras soluções similares, especialmente as que utilizam proxies onde todos os usuários entram e saem através do mesmo servidor, Tor passa seu tráfego através

de pelo menos três servidores diferentes antes de enviá-lo para o destino. Ele simplesmente retransmite o seu tráfego, completamente criptografado, através da rede Tor e tem que sair em outro lugar do mundo, completamente intacto. O cliente Tor é necessário para administrar a criptografia e o caminho escolhido pela rede.

Tor apresenta características de segurança relacionadas ao anonimato, a privacidade e a liberdade de expressão. Tor permite o **anonimato**, já que é usado por ativistas políticos, denunciantes, jornalistas, sobreviventes de violência doméstica e pessoas comuns ao redor do mundo que precisam para proteger suas identidades. Também garante a **privacidade**, uma vez que ajuda a esconder o endereço de IP de origem e evita o *fingerprinting* do navegador, tornando difícil que rastreadores on-line e até mesmo governos possam vigiar os usuários. Por fim, garante a **liberdade de expressão**, já que em países onde enormes áreas da Internet são bloqueados, as pessoas usam Tor para acessar a Web sem censura.

Defesas contra o Fingerprinting

Um *draft* do projeto Tor [Perry et al. 2013] enumera as defesas que o navegador Tor oferece contra as técnicas de *fingerprinting*. São elas:

1. **Plug-ins**: Além da sua própria funcionalidade (que pode ser desconhecida), plug-ins permitem que as técnicas de *fingerprinting* os descubram (através do objeto *Navigator*) e os explorem. Atualmente, todos os plug-ins são desativados no navegador Tor. No entanto, devido à popularidade do Flash, Tor permite que os usuários possam reativar Flash, mas objetos Flash são bloqueados por trás de uma barreira *click-to-play*, que fica disponível após o usuário habilitar o plug-in. Além disso, no Firefox, Tor define a preferência *plug-in.expose_full_path* para "false", para evitar o vazamento de informações com a instalação de plug-ins.

2. **Extração de imagem via HTML5 Canvas**: Os autores do documento acreditam que o HTML5 Canvas é a maior ameaça de *fingerprinting* a navegadores hoje em dia. Diferenças sutís na placa de vídeo, pacotes de fontes e até mesmo tipo de letra e versões de bibliotecas gráficas permitem produzir um *fingerprinting* simples e de alta entropia de um computador. Para reduzir esta ameaça, Tor recomenda que o navegador seja notificado antes de retornar dados de imagem válidos para as APIs do Canvas. Se o usuário não permitiu o site de acessar dados de imagem da tela, uma imagem em branco é devolvida para a API JavaScript.

3. **WebGL**: WebGL é passível de *fingerprinting* tanto através de informações que são expostas sobre o driver e suas otimizações quanto sobre sua performance. Por este motivo, Tor emprega uma estratégia similar ao dos plug-ins para WebGL. Primeiro, WebGL Canvas têm espaços reservados de *click-to-play* (fornecidos pelo NoScript) onde não executam até serem autorizados pelo usuário. Em segundo lugar, as informações do driver são ofuscadas através das preferências *webgl.disable-extensions* e *webgl.min_capability_mode*, que reduzem a informação fornecida pelas seguintes chamadas da API WebGL: **getParameter()**, **getSupportedExtensions()** e **getExtension()**.

4. **Fontes**: As fontes permitem identificação do navegador via Flash, plug-ins Java,

CSS e JavaScript [Eckersley 2010]. Como a solução para o problema das fontes seria o navegador possuir todas as fonte para todas as línguas, tipo de letra e estilo em uso no mundo (o que é praticamente impossível), Tor desativa os plug-ins, o que impede enumeração de fontes. Além disso, para limitar tanto o número de consultas de fontes CSS bem como o número total de fontes que podem ser usadas em um documento, duas preferências foram criadas e definidas, *browser.display.max_font_attempts* e *browser.display.max_font_count*.

5. **Resolução do desktop, Consultas a mídia CSS e sistema de cores**: Tanto o CSS quanto o JavaScript têm acesso a uma série de informações sobre a resolução de tela, sistema operacional, tamanho da barra de ferramentas, tamanho da barra de título, cores do tema do sistema, entre outros recursos de desktop que não são relevantes para renderização e servem apenas para fornecer informações *fingerprinting*. Tor tem várias estratégias implementadas para combater essas falhas.

6. **User-agent e cabeçalhos HTTP**: Todos os usuários do navegador Tor devem fornecer *user-agent* e cabeçalhos HTTP idênticos para um determinado tipo de solicitação. Para isso, Tor estabelece preferências semelhantes as já definas no Firefox para controlar os cabeçalhos *accept-Language* e *accept-charset*. Além disso, ele remove o acesso de scripts de conteúdo para *components.interfaces*, que podem ser usados para *fingerprinting* de sistema operacional e plataforma.

7. **Desempenho do JavaScript**: O desempenho do JavaScript já vem sendo usado [Mowery et al. 2011, Mulazzani et al. 2013] para identificar o motor de interpretação JavaScript e a CPU. Atualmente, embora existam várias possíveis soluções para o problema, somente uma solução está disponível no TOR, a desativação da propriedade "Navigation Timing" através da preferência *dom.enable_performance*.

8. **Implementação não uniformes da API HTML5**: Uma vez que pelo menos dois recursos HTML5 têm especificações diferentes da implementação nos principais fornecedores de sistemas operacionais (a API da bateria e a API de conexão de rede), Tor criou as preferências *dom.battery.enabled* e *dom.network.enabled*, no Firefox, para desativar essas APIs.

2.7.2. Extensões e plug-ins

2.7.2.1. FireGloves

FireGloves é um plug-in para Mozilla Firefox, o qual tem como finalidade principal impedir o rastreamento baseado em *fingerprinting* [FireGloves 2014]. Foi desenvolvido por uma equipe de pesquisadores da Universidade de Tecnologia de Budapeste com intuito de demonstrar que é possível impedir técnicas de *fingerprinting*. 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.

Recentemente, o plug-in passou por melhorias, mas testes realizados descobriram várias insuficiências, como aponta [Acar et al. 2013]. Por exemplo, em vez de confiar em valores para os elementos *offsetWidth* e *offsetHeight*, poderiam ser usados a largura e a altura do objeto *Rectangle*, retornado pelo método *getBoundingClientRect*, que fornece as dimensões do texto.

2.7.2.2. Ghostery

O Ghostery é uma extensão para os principais navegadores, 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 [Evidon 2014]. 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.

Se por um lado o Ghostery é uma ferramenta valiosa para a privacidade, e seu fabricante é experiente o suficiente para saber que ele não pode ganhar dinheiro apenas bloqueando Cookies de rastreamento, por outro lado, a empresa teve uma idéia inteligente: levantar os dados que os usuários fornecem sobre os bloqueios e vendê-los para as próprias empresas bloqueadas. Para isso, a extensão possui uma função de relatório, chamada "GhostRank", que envia informações anônimas sobre a tecnologia de coleta de dados que estão observando, bem como sua origem. Os dados desses relatórios são analisados e vendidos as empresas, a fim de ajudá-las a auditar e gerenciar seu relacionamento com ferramentas de marketing. Nenhuma das informações que compartilham é sobre seus usuários, nem é armazenada outra forma que poderia ser usada para rastreá-los. Esses mesmos dados também são enviados a Comissão do Comércio Federal (*Federal Trade Commission*[29]), que monitora as operações da indústria de publicidade. Por padrão, o Ghostrank fica desativado, o que significa que seus usuários podem usá-lo sem compartilhar nada com a empresa.

O fato mais importante é que a Evidon está levando a indústria de anúncio para a autoregulação, um esforço que está se tornando mais importante uma vez que o número de redes de publicidade continua a crescer. O Ghostery acrescenta cerca de 70 novas empresas seguidoras a cada três meses [Evidon 2014].

O funcionamento do Ghostery se dá através da verificação do código HTML em cada página Web que o usuário visita, onde é verificado a existência de tags ou "trackers" colocados por uma empresa que trabalha com esse site. 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.

[29]http://www.ftc.gov

2.7.2.3. Adblockplus

O Adblock Plus (ABP) é um filtro de conteúdo e um bloqueador de anúncios indesejados na Web [Adblock Plus 2014]. 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 Adblock Plus também faz o rastreamento de anúncios.

O funcionamento do Adblock Plus é apresentado na Figura 2.16. Na ilustração, quando um usuário faz a solicitação de um site, o ABP verifica em sua lista de filtros se no site existe algum tipo de anúncio que deva ser bloqueado. Se sim, ele realiza o bloqueio e apresenta no computador do usuário o site sem o anúncio indesejado.

Figura 2.16. Funcionamento do Adblock Plus. Fonte: [Adblock Plus 2014]

Assim como todos os softwares já apresentados até o momento, o Adblock Plus necessita ser ativado pelo usuário para que suas listas de filtros realizem o bloqueio nos sites. É importante relatar que ele possui duas listas de filtros habilitadas. A primeira, lista de *ad-blocking*, possui sites a serem bloqueados que foram selecionados com base no idioma do navegador. A segunda lista mantém os anúncios aceitáveis, ou seja, funciona como uma lista branca (*whitelist*). Contudo, o usuário é livre para desativar essas listas e adicionar ou criar outras [Adblock Plus 2014]. Outra vantagem do software é o fato de não coletar todos os dados do usuário e a maioria dos dados (por exemplo, os sites que o usuário visita) nunca é enviada para os servidores do fabricante [Adblock Plus 2014].

2.7.2.4. StopFingerprinting

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 [INRIA 2014]. Os dados coletados somente serão utilizados para fins de pesquisa dentro

INRIA e não serão 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.

2.7.2.5. Lightbeam

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 [Mozilla 2014]. 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.

A Figura 2.17 dá uma demonstração da visão do Lightbeam.

2.7.3. Do-Not-Track (DNT)

O cabeçalho *Do-Not-Track* (DNT) é uma proposta de campo adicional no cabeçalho do HTTP, através do qual é possível que uma aplicação Web ou um usuário solicite a desativação do seu rastreamento. Definido inicialmente em [Mayer et al. 2011], DNT está sendo padronizado pelo W3C com o nome "Tracking Preference Expression" [W3C 2014]. 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, [Eletronic Frontier Foundation 2014] 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. O DNT aceita três valores de configuração: (1) Caso o usuário não deseje ser rastreado (opt-out); (0) No caso do usuário autorizar ser rastreado (opt-in); (Nulo) - sem cabeçalho enviado - Se o usuário não manifestar uma preferência. O comportamento padrão exigido pela norma

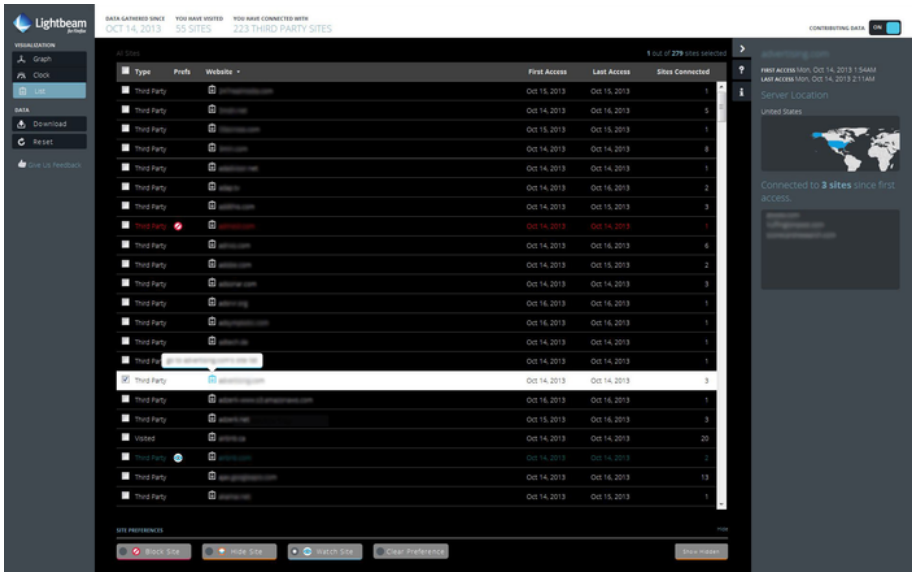

Figura 2.17. Gráfico de navegação do Lightbeam. Fonte: [Mozilla 2014]

é não enviar o cabeçalho, a menos que o usuário permita a configuração através do seu navegador ou sua escolha esteja implícita ao uso desse navegador específico.

Funcionamento

A Figura 2.18 ilustra o funcionamento do DNT.

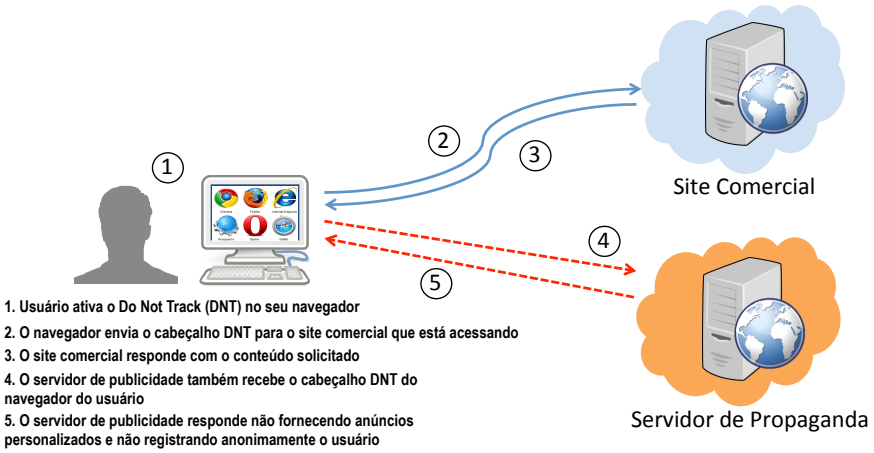

Figura 2.18. Funcionamento do DNT

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. Assim, quando o usuário visita um site de comércio eletrônico, ele informa ao servidor do site comercial que não quer ser rastreado. Por sua vez, o site comercial entende e responde a solicitação feita pelo usuário. Assim, quando os dados começam a chegar no navegador, ele também envia aos servidores de publicidade on-line que também não quer ser rastreado. Os servidores de publicidade aceitam o uso do DNT e enviam ao navegador do usuário anúncios não personalizados e não ativam mecanismos de identifi-

cação.

Embora não haja nenhum padrão acordado, de modo universal, sobre o que uma empresa deve fazer quando detecta um cabeçalho DNT, um grupo de trabalho internacional de defensores de políticas, especialistas técnicos e empresas está tentando criar uma interpretação consensual. Atualmente, várias empresas implementam a política Do Not Track em seus sites. BlueCava, DailyMail, Pinterest e Twitter são alguns exemplos [Future of Privacy Forum 2014].

2.8. Considerações Finais

As técnicas de *device fingerprinting* estão se tornando cada vez mais presentes e comuns na Web e mesmo assim os usuários não são informados sobre isso. Mesmo aqueles que estão cientes de que estão sendo monitorados, por exemplo, como uma medida de proteção contra fraudes, confiam, sem nenhum tipo de prova, que as informações coletadas não são ou serão utilizadas para outros fins. 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, 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 Capítulo procurou introduzir o leitor no universo do *fingerprinting* na Web. Em primeiro lugar, o problema da coleta de informações de usuários/maquinas, sem prévia autorização, foi contextualizado e foram apresentadas definições bem como uma discussão sobre privacidade também foi efetuada. Os navegadores e as tecnologias atuais, causas ou fontes da capacidade de se executar um *fingerprinting*, foram apresentadas, tendo suas características e sua potencialidade para *fingerprinting* discutidas e exemplificadas. Em seguida, de maneira didática, foi apresentado um simples roteiro de construção de um mecanismo para *device fingerprinting*. Como prova da conceito e comprovação da capacidade de obter dados via Web, o resultado de um *fingerprinting* usando o site noc.to foi ilustrada. Perto do fim, as soluções existentes (tanto acadêmicas quanto comerciais) para realização de um *fingerprinting* bem como as contramedidas foram discutidas. Para estimular ainda mais o leitor, serão discutidas algumas questões em aberto e, por fim, serão feitas as observações finais.

2.8.1. Pesquisas em aberto

Algumas questões sobre *device fingerprinting* que ainda estão em aberto serão discutidas a seguir:

- Hoje em dia, as principais defesas contra *device fingerprinting* são os plug-ins e extensões para navegadores, que permitem ao usuário não informar dados ou alterar valores que identificam seu navegador para um servidor. Contudo, os trabalhos tem mostrado que as técnicas mais recentes de *device fingerprinting* podem facilmente superar tais extensões. Isso porque os navegadores modernos são enormes pedaços de software, cada um com suas peculiaridades. Assim, melhorar a construção dos navegadores é uma forma de limitar o *device fingerprinting* e uma fonte

de trabalhos.

- É consenso que quanto melhor for o software do navegador, melhor será sua resposta a *fingerprinting*. Nesse sentido, uma solução simples pode ser barrar os scripts de *fingerprinting* toda vez que forem carregados nos navegadores, semelhante à maneira como funcionam os bloqueadores de anúncios. Mantendo uma *blacklist* de roteiros problemáticos, uma extensão *anti-fingerprinting* poderia detectar seu carregamento e proibir sua execução. Contudo, surgem dois desafios. O primeiro é manter a *blacklist* constantemente atualizada a fim de acompanhar as mudanças que *trackers* certamente fariam em resposta. Outro desafio é que não se sabe se o carregamento de scripts de *fingerprinting* são necessários para o funcionamento de determinados sites. Mesmo que isso não seja necessário agora, os sites poderiam ser alteradas para recusar o carregamento de suas páginas a menos que os scripts *fingerprinting* estejam presentes e operacionais, o que desencorajaria as pessoas de tentar interferir com eles.

- Dado que a publicidade é a indústria número 1 da Web e que o rastreamento é um componente crucial dessa indústria, o mecanismos de geração de perfil do usuário e as técnicas de *device fingerprinting* vão existir por um bom tempo. Entretanto, regulamentos mais rigorosos e contramedidas mais eficazes podem desacelerar essa indústria e impedir piores e maiores abusos.

- Já existem empresas que oferecem navegação baseada em nuvem. Entretanto, não está claro se os navegadores que estão expostos a potenciais *fingerprinters* realmente operam na nuvem. Ainda assim, não há nenhuma razão para pensar que um sistema de prevenção de *device fingerprinting* para um navegador em nuvem não possa ser projetado.

2.8.2. Observações Finais

Atualmente, não existe uma lâmpada mágica que forneça a resposta definitiva contra *device fingerprinting*, mas algumas ações podem e devem ser tomadas:

- Aumentar o número de pesquisas nessa área visando, inicialmente, resolver problemas específicos. O apoio de agências governamentais, órgãos de fomento à pesquisa e a própria iniciativa privada devem servir de fontes financiadoras.

- Estimular a realização de eventos, workshops, conferências sobre o tema. Tais encontros servem como ponto de partida para troca de conhecimentos e o surgimento de novas parcerias.

- Desenvolver um modelo jurídico de alcance global para definir a privacidade na Web. Este trabalho deve ser conduzido por especialistas de modo a garantir da melhor forma a flexibilidade da Internet.

- Aumentar o desenvolvimento e uso de soluções contra *device fingerprinting*. Mesmo que não encerrem o problema, podem diminuir a incidência de problemas de privacidade.

- Por fim, educar os usuários Internet para torná-los mais conscientes dos riscos existentes para seus computadores e sistemas.

Referências

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

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

[Adblock Plus 2014] Adblock Plus (2014). Adblock plus surf the web without annoying ads. `https://adblockplus.org/`. [Online, Acessado 25/09/2014].

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

[Bos et al. 2011] Bos, B., Celik, T., Hickson, I., and Lie, H. W. (2011). Cascading style sheets level 2 revision 1 (css 2.1) specification. W3c recommendation, W3C. `http://www.w3.org/TR/CCS2`.

[CCEE 2012] CCEE (2012). Cross-Browser Performance. `http://ccee.org.br/ccee/documentos/CCEE_056670`. [Online, Acessado 15/08/2014].

[Contextis 2011] Contextis (2011). WebGL - More WebGL Security Flaws. `http://www.contextis.com/resources/blog/webgl-more-webgl-security-flaws/`. [Online, Acessado 15/09/2014].

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

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

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

[ECMA International 2011] ECMA International (2011). *Standard ECMA-262 - ECMAScript Language Specification*. 5.1 edition. `http://www.ecma-international.org/publications/standards/Ecma-262.htm`.

[Eletronic Frontier Foundation 2014] Eletronic Frontier Foundation (2014). Do not track. `https://www.eff.org/issues/do-not-track`.

[Evidon 2014] Evidon (2014). Ghostery. http://www.ghostery.com/. [Online, Acessado 20/09/2014].

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

[FireGloves 2014] FireGloves (2014). Firegloves - cross-browser fingerprinting test 2.0. http://fingerprint.pet-portal.eu/?menu=6. [Online, Acessado 21/09/2014].

[Forshaw 2011] Forshaw, J. (2011). WebGL - A New Dimension for Browser Exploitation. http://www.contextis.com/resources/blog/webgl-new-dimension-browser-exploitation/. [Online, Acessado 15/09/2014].

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

[Future of Privacy Forum 2014] Future of Privacy Forum, F. (2014). About do not track. http://allaboutdnt.com. [Online, Acessado 20/09/2014].

[Giura et al. 2014] Giura, P., Murynets, I., Piqueras Jover, R., and Vahlis, Y. (2014). 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, pages 333–344, New York, NY, USA. ACM.

[Goodin 2013] Goodin, D. (2013). Stealthy technique fingerprints smartphones by measuring users movements. http://arstechnica.com/security/2013/10/stealthy-technique-fingerprints-smartphones-by-measuring-users-movements/. [Online, Acessado 19/09/2014].

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

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

[INRIA 2014] INRIA (2014). Stopfingerprinting - how trackable is your browser in a long term? https://stopfingerprinting.inria.fr. [Online, Acessado 25/09/2014].

[Kamkar 2010] Kamkar, S. (2010). Evercookie. http://samy.pl/evercookie/. [Online, Acessado 15/08/2014].

[Kirk 2014] Kirk, J. (2014). 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. [Online, Acessado 21/09/2014].

[Kristol and Montulli 1997] Kristol, D. and Montulli, L. (1997). HTTP State Management Mechanism. RFC 2109 (Proposed Standard). http://www.ietf.org/rfc/rfc2109.txt.

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

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

[McDonald and Cranor 2011] McDonald, A. M. and Cranor, L. F. (2011). A survey of the use of adobe flash local share objects to respawn http cookies. *Journal of Information Policy*, 7(3):639–687.

[Microsoft 2014] Microsoft (2014). Microsoft silverlight 5. http://www.microsoft.com/silverlight/. [Online, Acessado 21/09/2014].

[Modernizr 2014] Modernizr (2014). Modernizr: the feature detection library for html5/css3. http://modernizr.com. [Online, Acessado 24/09/2014].

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

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

[Mozilla 2014] Mozilla (2014). Lightbeam shine a light on whos watching you. https://www.mozilla.org/en-US/lightbeam/. [Online, Acessado 15/08/2014].

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

[Nikiforakis et al. 2014] Nikiforakis, N., Acar, G., and Saelinger, D. (2014). Browse at your own risk. *Spectrum, IEEE*, 51(8):30–35.

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

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

[Perry et al. 2013] Perry, M., Clark, E., and Murdoch, S. (2013). The design and implementation of the tor browser [draft]. https://www.torproject.org/projects/torbrowser/design/. [Online, Acessado 15/09/2014].

[Shankland 2011] Shankland, S. (2011). Microsoft declares webgl 'harmful' to security. http://www.cnet.com/news/microsoft-declares-webgl-harmful-to-security/. [Online, Acessado 24/09/2014].

[Slivinski 2011] Slivinski, A. R. (2011). Desenvolvimento de uma metodologia para avaliação de desempenho gráfico 3d de plataformas com suporte ao webgl. Technical report, Unioeste - Universidade Estadual do Oeste do Paraná.

[Soltani et al. 2010] Soltani, A., Canty, S., Mayo, Q., Thomas, L., and Hoofnagle, C. J. (2010). Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management*. AAAI.

[StatCounter 2014] StatCounter (2014). Statcounter globalstats. http://gs.statcounter.com. [Online, Acessado 15/09/2014].

[Tanner 2013] Tanner, A. (2013). The web cookie is dying. here's the creepier technology that comes next. http://www.forbes.com/sites/adamtanner/2013/06/17/the-web-cookie-is-dying-heres-the-creepier-technology-that-comes-next/. [Online, Acessado 20/09/2014].

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

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

[W3C 2014] W3C (2014). Fingerprinting guidance for web specification authors. http://w3c.github.io/fingerprinting-guidance/.

[W3C 2014] W3C (2014). Html5: a vocabulary and associated apis for html and xhtml. http://www.w3.org/TR/html5/.

[W3C 2014] W3C (2014). Tracking preference expression (dnt). W3c working draft, W3C. http://www.w3.org/TR/tracking-dnt/.

[Weinberger 2011] Weinberger, A. (2011). The impact of cookie deletion on site-server and ad-server metrics in australia. http://www.comscore.com/Insights/Presentations-and-Whitepapers/2011/The-Impact-of-Cookie-Deletion-on-Site-Server-and-Ad-Server-Metrics-in-Australia-An-Empirical-comScore-Study.

