LETTY: Uma implementação de *Website Fingerprinting*

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

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

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

Abstract. *Website Fingerprinting techniques are capable of identify, with high probability, uniquely a device and consequently its user, by extracting device-related characteristics. Normally, those techniques harms user's privacy on the Web and the reason for this is that the Websites which embed Website Fingerprinting don't warn users of its existence nor of its functions. However, these techniques are used for authentication and validation purposes. Controversially, they can be used for malicious purposes (user activity tracking). With the goal to show that it is easy and even simple, this work has the objective to implement a Website Fingerprinting with JavaScript and to evaluate the real world possibilities of identifying uniquely a large number of devices.*

Resumo. *As técnicas de Website Fingerprinting são capazes de identificar, com alta probabilidade, unicamente um dispositivo e consequentemente seu usuário, através da extração de dados relacionados ao dispositivo. Normalmente, tais técnicas ferem a privacidade dos usuários na Web ao não avisar ou deixar claro a sua existência ou função. Contudo, elas são usadas para fins de autenticação e validação de usuários. Por outro lado, também podem ser usadas para fins maliciosos (rastrear atividades dos usuários). Como forma de mostrar que é fácil e até mesmo simples, este trabalho tem como objetivo implementar um Website fingerprinting usando códigos em JavaScript e avaliando a real possibilidade de se identificar unicamente um grande número de dispositivos.*

1. Introdução e Motivação

A Web tornou-se o principal meio para a realização de diversas atividades, graças a sua capacidade de oferecer várias aplicações e serviços, tais como pesquisas, compras, pagamentos online, interação com outras pessoas através de redes sociais, dentre outras [Khademi 2014].

Contudo, essa diversidade tem permitido aos interessados (empresas de propaganda, segurança digital e até mesmo atacantes) utilizar características intrínsecas e diferenças de implementação para obter informações relacionadas ao navegador e ao dispositivo (sistemas operacionais e listas de extensões e plugins, por exemplo) e, consequentemente, identificar o usuário por trás da máquina. Essa identificação é feita com o uso de técnicas de *website fingerprinting*, um conjunto de métodos e algoritmos capazes de identificar um usuário/navegador, com base nas informações do dispositivo [Eckersley 2010].

Em linhas gerais, *website fingerprinting* emprega um conjunto amplo de tecnologias e técnicas usadas para identificar (ou reidentificar) um usuário ou um dispositivo, através de um conjunto de configurações, atributos (tamanho da tela do dispositivo,

versões de software instalado, entre muitos outros) e outras características observáveis durante as comunicações [Saraiva et al. 2014]. A questão é: o quão fácil é obter informações sobre o navegador e o dispositivo de um usuário, a fim de gerar uma identificação única? Assim, o objetivo deste artigo é desenvolver um mecanismo de *website fingerprinting* aplicando técnicas simples, mas capazes de identificar usuários na Internet, comprovando que métodos e atributos específicos do *JavaScript* podem ser usados para implementar tal mecanismo. Para tanto, um protótipo funcional denominado LETTY (*LeT me idenTify You*) foi implementado e aplicado na identificação de usuários. Os resultados mostram que os atributos mais comuns passíveis de coleta em um dispositivo tem a capacidade de gerar um identificador.

O restante do artigo é organizado como segue: na Seção 2 é apresentada uma breve descrição sobre *website fingerprinting*, incluindo seu funcionamento e tecnologias atualmente utilizadas. Na Seção 3 é descrito o projeto e a implementação da LETTY, explicando as tecnologias utilizadas e a dinâmica de funcionamento. Na Seção 4 são apresentados os resultados dos experimentos realizados com a ferramenta. Por fim, a Seção 5 traz uma visão geral do trabalho e melhorias futuras que podem ser feitas.

2. *Website Fingerprinting*

Website fingerprinting é o processo em que as informações do dispositivo e do navegador são coletadas, a princípio, para autenticação e identificação do usuário [Mowery et al. 2011, Khademi 2014, Eckersley 2010], para evitar fraudes de transações [Nikiforakis et al. 2013], para rastrear usuários [Nikiforakis and Acar 2014], para evitar roubo de sessão [Unger et al. 2013], dentre outras aplicações. Também é possível fazer *fingerprinting* apenas do navegador, obtendo informações estritamente relacionadas ao mesmo (versão, família, etc) [Mulazzani et al. 2013] e assim ajustar os conteúdos da página que serão exibidos, a fim de proporcionar uma boa experiência de navegação.

De acordo com a RFC 6973 [Cooper et al. 2013], *fingerprinting* é o processo no qual o observador ou atacante, com uma alta probabilidade, identifica unicamente um dispositivo ou aplicação, baseado na comunicação que ocorre entre os elementos que estão sendo observados.

Alguns autores na literatura consideram *website fingerprinting* como o método usado para descobrir o navegador, sua família e versão [Mulazzani et al. 2013]. Já outros consideram o termo como técnicas e tecnologias utilizadas para identificar um usuário [Eckersley 2010, Khademi 2014]. Neste trabalho, *website fingerprinting* é considerado como um conjunto de técnicas e tecnologias que são capazes de extrair informações de um navegador e usar essas informações para identificar tanto o dispositivo quanto o usuário.

2.1. Classificação

A RFC 6973 [Cooper et al. 2013] preconiza a existência de três tipos de *fingerprinting*: passivo, ativo e *cookie-like*. O **passivo** é baseado nas características observáveis das solicitações Web (cabeçalhos HTTP, endereço IP e outras informações do nível de rede), sem utilização de qualquer código executando no lado do cliente. O **ativo** utiliza códigos dinâmicos (JavaScript, ActionScript ou por qualquer outro tipo de código) no lado do cliente para observar características sobre o navegador. Dentre as características observáveis podem-se destacar a lista de fontes disponível no navegador, o tamanho da

tela, lista de *plugins*, padrões de renderização de gráficos, entre outros. O terceiro e último tipo é o ***cookie-like***, onde um código ou aplicativo é instalado no dispositivo do usuário, permitindo sua (re)identificação e a geração de inferências sobre o mesmo.

Neste trabalho, o tipo **ativo** foi utilizado na construção da ferramenta LETTY, pois as observações das características são realizadas no lado do cliente, com o uso da linguagem JavaScript.

2.2. Tecnologias

Para criar um mecanismo ou algoritmo de *website fingerprinting* é necessário utilizar alguma tecnologia Web disponível. Dentre as comumente empregadas estão:

- JavaScript, que tem acesso fácil e direto a diferentes APIs que fornecem informações relevantes sobre navegador e dispositivo em que o navegador está sendo executado (por exemplo, lista de *plugins* e sistema operacional) [Khademi 2014].
- Canvas, que é empregado para desenhar elementos 2D, mas que também permite a recuperação do conteúdo desenhado e a consequente identificação do navegador e do sistema operacional [Khademi 2014].
- Flash, que assim como JavaScript, possui métodos que permitem o acesso a várias informações que estão estritamente relacionadas ao sistema em que o navegador está sendo executado. Por exemplo, pode ser obtido o nome do sistema operacional, arquitetura da CPU, número de DPI (*Dots Per Inch*) da tela, dentre outras características [Khademi 2014].
- CSS, que se vale das diferentes versões e implementações dos navegadores para avaliar diferenças na *engine* de *layout* a fim de identificar o navegador [Unger et al. 2013].
- Cabeçalho HTTP, uma vez que existem uma série de informações que caracterizam um navegador e são suficientes para sua identificação [Unger et al. 2013].

3. Projeto e Implementação

Esta Seção apresenta os atributos coletados e a visão geral do projeto, incluindo detalhes da implementação da ferramenta LETTY.

3.1. Objetos e Atributos

Os dois principais objetos empregados na identificação do navegador são o *screen* e o *navigator*. O primeiro possui informações sobre as configurações de tela em que as páginas estão sendo renderizadas pelo navegador, enquanto o segundo possui informações sobre o ambiente em que o navegador está sendo executado, bem como suas propriedades.

Os atributos obtidos através dos objetos *screen* e *navigator* são exibidos nas Tabelas 1 e 2, respectivamente.

3.2. Visão geral

A ferramenta LETTY foi projetada de forma a ser integrada a qualquer *website* com bastante facilidade. Por isso, sua implementação fez uso da linguagem JavaScript, que permite coletar informações de ambos objetos e armazená-las em qualquer estrutura de dados. Além disso, segundo [Khademi 2014], o *fingerprint* gerado com o uso do JavaScript

Tabela 1. Atributos do objeto *screen*.

Propriedade	Descrição
colorDepth	Contém a profundidade de cores utilizadas para renderizar imagens na tela do dispositivo em que o navegador está sendo executado.
pixelDepth	Contém a resolução de cores da tela do dispositivo em que o navegador está sendo executado.
width	Contém a largura da tela do dispositivo em que o navegador está sendo executado.
height	Contém a altura da tela do dispositivo em que o navegador está sendo executado.

Tabela 2. Atributos do objeto *navigator*.

Propriedade	Descrição
userAgent	Contém informações sobre nome, versão e plataforma do navegador.
product	Contém o nome da *engine* do navegador.
productSub	Contém informação sobre o número da versão de desenvolvimento da *engine* do navegador.
cookieEnabled	Retorna *true* ou *false* se os *cookies* são permitidos ou se não são permitidos.
vendor	Informa quem é o desenvolvedor do navegador.
platform	Contém informação sobre a plataforma para a qual o navegador foi compilado.
language	Contém informação sobre a linguagem atual do navegador.
languages	Contém informação sobre as linguagens disponíveis do navegador.
javaEnabled	Informa se o navegador possui ou não o Java ativado.
appName	Contém o nome do navegador.
appCodeName	Contém o codinome do navegador.
appVersion	Contém a versão do navegador.
oscpu	Contem a arquitetura do processador do computador em que o navegador está sendo executado.
maxTouchPoints	Contém o número máximo de toques suportados pela tela do dispositivo em que o navegador está sendo executado.
plugins	Contém os *plugins* instalados no navegador.
mimeTypes	Contém a lista de tipos MIME suportados pelo navegador web.

possui alta entropia (podendo chegar até 9.97 bits). Quanto maior a entropia, maiores as chances de um dado dispositivo ser identificado unicamente. A Figura 1 ilustra a dinâmica de funcionamento da LETTY.

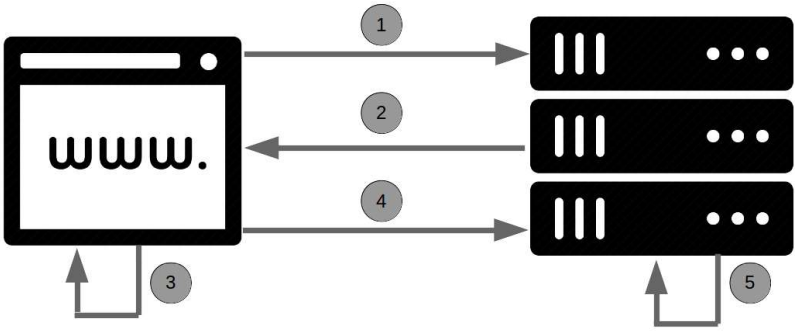

Figura 1. *Workflow* da LETTY.

No **passo 1**, o cliente (usuário) solicita acesso a uma página Web (contendo a LETTY). Como resposta (**passo 2**), o servidor Web envia a página solicitada (conteúdo), incluindo o código de *fingerpinting*. Ao chegar no navegador (**passo 3**) do cliente, a LETTY começa a executar, coletando os dados disponíveis no navegador. Primeiramente, os atributos do objeto *navigator* ($userAgent$, $product$, $productSub$, $cookieEnabled$, $vendor$, $platform$, $language$, $languages$, $javaEnabled$, $appName$, $appCodeName$, $appVersion$, $oscpu$ e $maxTouchPoints$) são coletados. Em seguida, é a vez dos atributos do objeto *screen* ($colorDepth$, $pixelDepth$, $width$ e $heigth$). De-

pois os *plugins* presentes no navegador ($navigator.plugins$) e os *mimetypes* suportados ($navigator.mimeTypes$) são coletados. Embora ambos estejam ligados ao objeto *navigator*, sua coleta é diferenciada visto que ambos são representados por vetores de atributos (e não características únicas como os demais).

No **passo 4**, a LETTY devolve ao servidor os dados coletados. No servidor, os dados são processados e a chave de identificação única do usuário/dispositivo é gerada (**passo 5**). Para gerar a chave, uma função de *hashing*, da biblioteca Cripto.js[1], foi utilizada. Embora se saiba que podem ocorrer colisões em tais funções, as chances dependendo da função escolhida podem ser extremamente baixas. Para tanto, foi empregada uma função *hash* MD5 de 128 bits, cuja probabilidade de colisão é estimada em ocorrer quando existirem 2^{64} entradas [Krawczyk et al. 1997]. Vale ressaltar que um banco de dados pode ser empregado para guardar os dados coletados dos clientes.

4. Avaliação e Resultados Experimentais

Para comprovar sua efetividade, a LETTY foi incorporada em uma página Web hospedada em um servidor da Universidade (https://200.17.49.135/pesquisa/letty). Os experimentos foram realizados durante 4 semanas, entre os dias 6 de outubro e 9 de novembro de 2015. No final, foram realizados 230 acessos à página com 119 dispositivos identificados unicamente.

Em linhas gerais, quando a página é acessada, é exibida uma mensagem explicando rapidamente o experimento e o propósito do mesmo. A LETTY só realiza o *fingerprinting* do usuário caso o mesmo concorde em participar do experimento. Uma vez aceito, os dados do navegador são recolhidos, enviados ao servidor (onde o identificador é gerado) e os valores são então exibidos para o usuário que visitou a página. A Figura 2 ilustra um resultado final exibido para o usuário.

4.1. Resultados

Antes de apresentar de fato o que foi descoberto, é importante ressaltar que como alguns atributos sempre apresentam os mesmos valores, seus resultados não são apresentados. Por exemplo, todos os navegadores que acessaram a página permitem o uso de *cookies* (*cookieEnabled* = *yes*) e também do Java (*javaEnabled* = yes). *appCodeName* representa o codinome do navegador e, por questões de compatibilidade, todos os navegadores modernos retornam Mozilla [W3Schools 2016].

Plataformas e navegadores

A Tabela 3 ilustra as plataformas e os navegadores dos dispositivos que foram usados para acessar a página Web. Todos os dados coletados foram obtidos através dos métodos *userAgent* e *appVersion*, do objeto *navigator*.

Percebe-se na Tabela 3 que uma variedade de navegadores foram utilizados para acessar a página da ferramenta LETTY. Segundo os dados apresentados, o navegador Chrome é o mais prevalente (63 aparições), seguido do Firefox (27 aparições) e Safari (10 aparições). Sobre o navegador Chrome, foram encontradas tanto versões mais recentes (50, 51 e 52) quanto versões mais antigas (36, 41, 42 e 43), todas com diferentes *releases*.

[1]https://www.npmjs.com/package/crypto-js

Attribute	Content
User Agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.134 Safari/537.36
Product	Gecko
Product Sub	20030107
Cookie Enabled	true
Vendor	Google Inc.
Platform	Linux x86_64
Language	pt-BR
Languages	pt-BR,pt,en-US,en
Java Enabled	function javaEnabled() { [native code] }
App Name	Netscape
App CodeName	Mozilla
App Version	5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.134 Safari/537.36
Oscpu	undefined
Mime Types	application/pdf,application/x-shockwave-flash,application/futuresplash,application/vnd.chromium.remoting-viewer,application/x-ppapi-widevine-cdm,application/googletalk,application/o1d,application/x-nacl,application/x-pnacl,application/x-google-chrome-pdf
Device	desktop
Key	8d13d3322e556fb2669ffe220537da05

Figura 2. Exemplo de resposta dada ao usuário sobre os dados coletados com a LETTY.

Tabela 3. Plataformas e Navegadores

Plataformas x Navegadores	Chrome	Firefox	Safari	Opera Coast	Samsung Browser	Blackberry Browser	LG Netcast 3	Internet Explorer	Edge	Epiphany	Miui Browser	Chromium	Amazon Silk 3	Desconhecido
Windows 10	13	9	0	0	0	0	0	1	3	0	0	0	0	0
Widows 8.x	3	1	0	0	0	0	0	0	0	0	0	0	0	0
Windows 7	9	3	0	0	0	0	0	1	0	0	0	0	0	0
Windows Vista	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Linux	11	8	0	0	0	0	0	0	0	0	0	1	0	0
Android 6.x	6	0	0	0	0	0	0	0	0	0	0	0	0	0
Android 5.x	10	0	0	0	1	0	0	0	0	0	0	0	0	0
Android 4.x	5	2	0	0	1	0	0	0	0	0	1	0	1	1
Android 2.x	0	0	0	0	1	0	0	0	0	0	0	0	0	0
iOS 9.3	1	1	4	1	0	0	0	0	0	0	0	0	0	0
iOS 9.0	0	1	2	0	0	0	0	0	0	0	0	0	0	0
Mac OS X	4	2	4	0	0	0	0	0	0	0	0	0	0	0
Blackberry	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Raspberry Pi	0	0	0	0	0	0	0	0	0	1	0	0	0	0
LG WebOS	0	0	0	0	0	0	1	0	0	0	0	0	0	0
TOTAL	**63**	**27**	**10**	**1**	**3**	**1**	**1**	**2**	**3**	**1**	**1**	**1**	**1**	**1**

Além dos dados mais frequentes, a Tabela 3 também apresenta plataformas e navegadores não tão usuais (BlackBerry, Raspberry Pi, Opera Cost e Amazon Silk 3). Essas características peculiares destacam ainda mais o dispositivo que passou pelo processo

de *fingerprinting*. Um exemplo mais visual é apresentado com a *string* do *userAgent* retornado pelo navegador Epiphany, utilizado pelo Raspberry Pi: **Mozilla/5.0 (Macintosh; ARM Mac OS X) AppleWebKit/538.15 (KHTML, like Gecko) Safari/538.15 Version/6.0 Raspbian/8.0 (1:3.8.2.0-0rpi27rpi1g) Epiphany/3.8.2**.

A primeira vista, pode-se pensar que o acesso foi oriundo de algum dispositivo rodando Mac OS ou iOS. No entanto, Epiphany é um navegador que não está disponível para esses sistemas.

Por fim, embora o experimento tenha coletado 119 *fingerprints* no total, apenas 116 são exibidos na Tabela, uma vez que três dos *fingerprints* coletados pertencem a robôs indexadores de página. Além disso, um único navegador foi classificado como desconhecido, pois os métodos empregados (*userAgent* e *appVersion*) não foram capazes de identificá-lo. Explicando melhor, a *string* retornada pelo método *userAgent* o apresenta como sendo Safari, mas o acesso foi feito a partir de um dispositivo Android. Embora não seja conclusivo, pode-se atribuir tal comportamento a alguma solução de segurança capaz de mudar a *string* do *userAgent*, retornando valores diferentes a cada requisição.

Resolução

Na coleta realizada com a LETTY, foram encontradas diferentes resoluções de tela dos dispositivos que acessaram a página, conforme ilustrado na Figura 3. Vale ressaltar que a resolução retornada representa o que é exibido pelo navegador, não incluindo resoluções de fotos e vídeos.

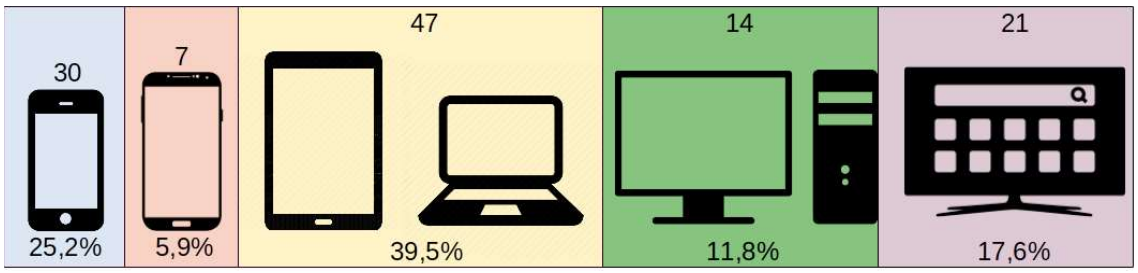

Figura 3. Quantidade, dimensão de tela e percentagem de cada tipo de dispositivo.

Foram identificados 30 dispositivos com telas de pequenas dimensões, variando de 320x534 até 384x640, representando 25,2% do total encontrado. Também foram identificados 7 dispositivos com dimensões entre 412x732 e 800x1270, perfazendo 5,9%. Esses 37 dispositivos são telefones celulares e smartphones executando Android, iOS ou Blackberry OS.

Os *notebooks* e *tablets* formam a maior faixa dos identificados, totalizando 47 dispositivos (39,5%) e com variação de tela indo de 1000x600 até 1366x768. Neste intervalo se enquadra o *smartphone* BlackBerry apresentado na seção de plataformas e navegadores. Os *desktops* identificados representam 11,8% dos dispositivos (14), apresentando resoluções entre 1438x808 e 1824x984.

Por fim, a última categoria é composta por *SmartTVs* e *desktops* ou *notebooks* de alta resolução ou usando mais de uma tela. Foram identificados 21 dispositivos com resolução entre 1916x979 e 1920x1200. Nesta última categoria também enquadram-se

os robôs indexadores que visitaram a página da LETTY (não considerados na seção de plataformas e navegadores).

Vale ressaltar que os valores das resoluções foram obtidos através dos métodos *width* e *height* do objeto *screen*.

Plugins

Além de obter dados sobre as plataformas, navegadores e resoluções, foi possível também obter a lista de *plugins* que os usuários utilizam em seus navegadores. Os *plugins* coletados durante o experimento estão ilustrados na Figura 4. Quanto maior for o tamanho da fonte, maior o número de ocorrências do *plugin* no conjunto dos resultados analisados.

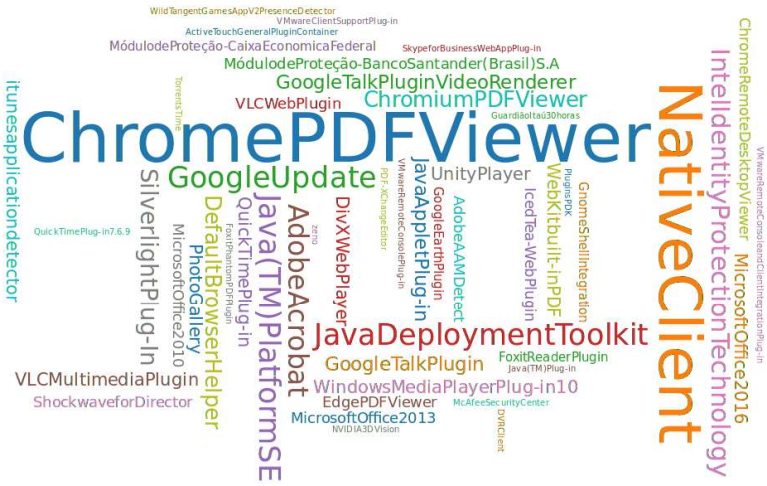

Figura 4. Nuvem de palavras com a lista de *Plugins*.

Obviamente, percebe-se que na Figura 4 existe uma diversidade de *plugins* nos navegadores modernos. Contudo, o preocupante é a questão da privacidade e segurança. Embora, a primeira vista, saber os *plugins* que o navegador de um usuário suporta não represente qualquer ameaça, o processo de coleta da LETTY foi capaz de encontrar *plugins* específicos relacionados à *Internet banking*. Foram encontrados os módulos de proteção dos bancos Caixa Econômica Federal e Santander.

Outros Resultados

Uma vez que a função da LETTY é testar a capacidade de coleta de informações sobre o navegador e o dispositivo, foi realizado um experimento para avaliar a resposta da LETTY em um ambiente com máquinas clonadas (máquinas com as mesmas configurações de *hardware* e contendo os mesmos *softwares* instalados de uma única vez). Neste experimento foi realizado em um laboratório contendo 16 máquinas idênticas.

Como resultado, apenas a primeira máquina a acessar a página contendo a LETTY foi identificada e teve seu identificador gerado. Todas as outras 15 máquinas foram vistas como sendo acessos repetidos da primeira, demonstrando que apesar de representativos, os atributos empregados no *fingerprinting* precisam ser melhorados.

5. Conclusões

Fingerprinting é um conjunto de técnicas que podem ser usadas tanto para fins benignos (autenticação, prover melhor experiência de uso, etc) quanto maliciosos (invasão de privacidade, explorar vulnerabilidades, dentre outros), uma vez que permitem obter, com certo nível de facilidade e relativa precisão, informações sobre o dispositivo e o usuário. Além disso, técnicas de *fingerprinting* voltadas a Web não precisam, a princípio, armazenar nenhum tipo de informação no dispositivo do usuário, diferentemente dos *cookies*. Grande parte dessa facilidade é devida a disponibilidade de APIs capazes de retornar dados relacionados à máquina do usuário. Dentre as tecnologias utilizadas em *fingerprinting*, a linguagem JavaScript é sem dúvida a mais usada por ser capaz de fornecer um variado número de informações (versão do navegador, lista de *plugins*, resolução de tela, número máximo de toques suportados pela tela, sistema operacional, entre outras) e assim identificar unicamente um dispositivo.

Este trabalho implementou um *fingerprinting* através dos objetos *navigator* e *screen*, utilizando JavaScript, e provou que é possível tirar vantagem de informações que todo navegador precisa responder, e ainda mais, mostrou que é simples obter tais dados e consequentemente identificar unicamente um dispositivo. A ferramenta desenvolvida neste trabalho (LETTY) foi construída com média quantidade de esforço, sendo capaz de descobrir características de todos os dispositivos que passaram pelo processo de *fingerprinting*. Além disso, um identificador único para cada dispositivo foi gerado e uma base com dados coletados também.

Contudo, este trabalho também mostrou que existem casos que precisam ser tratados, pois a identificação do usuário pode mudar por diversos fatores (presença de duas telas, atualização na versão do navegador, uso de um mecanismo de segurança, entre outros).

5.1. Trabalhos futuros

Embora o *fingerprinting* implementado neste trabalho tenha servido aos seus propósitos, é possível melhorá-lo para deixar a solução mais robusta. Um primeiro melhoramento é tentar descobrir se houve mudanças no dispositivo. Já foi provado em [Eckersley 2010] que é possível criar um algoritmo para verificar se um dispositivo com um novo *fingerprint* na verdade é um antigo dispositivo que teve a sua chave mudada por algum motivo. No entanto, achar uma solução ótima ainda é um trabalho em aberto.

Um segundo trabalho é de fato escolher atributos mais robustos e que retornem informações únicas para geração do identificador. O *fingerprinting* implementado neste trabalho utiliza o atributo *userAgent*, que pode ser mudado facilmente com o uso de extensões, alterando, assim, sua identificação. Por exemplo, [Nakibly et al. 2015] propuseram um *fingerprinting* baseado nos dispositivos de entrada e saída de áudio. A justificativa dos autores é que o microfone e fone possuem frequências diferentes, dependendo de como são projetados. Essas lacunas, combinadas a alguma inconsistência durante o processo de fabricação, por menor que seja, contribuem para identificar e rastrear um usuário e suas atividades.

5.2. Disponibilidade da ferramenta

O repositório da LETTY está disponível no seguinte link: https://github.com/Jordan-Queiroz/Letty.git.

5.3. Agradecimentos

Este trabalho teve apoio do Programa de Capacitação de Recursos Humanos em Tecnologias da Informação e Computação e em Sistemas e Aplicativos para Plataformas Tecnológicas Portáteis, Móveis e Distribuídas; número 930000, financiado pela Samsung Eletrônica da Amazônia.

Referências

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

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.

Khademi, A. F. (2014). Browser fingerprinting: Analysis, detection, and prevention at runtime. Master's thesis, Queen's University.

Krawczyk, H., Bellare, M., and Canetti, R. (1997). Hmac: Keyed-hashing for message authentication. RFC 2104, RFC Editor.

Mowery, K., Bogenreif, D., Yilek, S., and Shacham, H. (2011). Fingerprinting information in JavaScript implementations. In Wang, H., editor, *Proceedings of W2SP 2011*. IEEE Computer Society.

Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., Weippl, E., and Wien, F. (2013). Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, volume 5.

Nakibly, G., Shelef, G., and Yudilevich, S. (2015). Hardware fingerprinting using HTML5. *CoRR*, abs/1503.01408.

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

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.

Saraiva, A., Elleres, P., Carneiro, G., and Feitosa, E. (2014). Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas. In dos Santos, A., van de Graaf, J., Nogueira, J. M., and Oliveira, L. B., editors, *Livro de Minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg2014*, chapter 2, pages 49–98. SBC.

Unger, T., Mulazzani, M., Frühwirt, 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. IEEE.

W3Schools (2016). Navigator appcodename property. `http://www.w3schools.com/jsref/prop_nav_appcodename.asp`. Acesso 20/06/2016.

