Pós-Graduação em Ciência da Computação

EDGAR JOSÉ STELLO JUNIOR

DETECÇÃO DA UTILIZAÇÃO DO MECANISMO DE CANVAS FINGERPRINTING

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2017

EDGAR JOSÉ STELLO JUNIOR

DETECÇÃO DA UTILIZAÇÃO DO MECANISMO DE CANVAS FINGERPRINTING

Este trabalho foi apresentado à Pós-Graduação em Ciências da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre Profissional em Ciências da Computação.

ORIENTADOR: **Prof. Dr. Ruy José Guerra Barretto de Queiroz**

RECIFE
2017

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S824d Stello Junior, Edgar José
Detecção da utilização do mecanismo de Canvas Fingerprinting / Edgar José Stello Junior. – 2017.
70 f.:il., fig.

Orientador: Ruy José Guerra Barreto de Queiroz.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2017.
Inclui referências e apêndice.

1. Ciência da computação. 2. Segurança da informação. I. Queiroz, Ruy José Guerra Barreto de (orientador). II. Título.

004 CDD (23. ed.) UFPE- MEI 2017-186

EDGAR JOSÉ STELLO JUNIOR

DETECÇÃO DA UTILIZAÇÃO DO MECANISMO DE CANVAS FINGERPRINTING

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre Profissional em 25 de abril de 2017.

Aprovado em: 25/04/2017

BANCA EXAMINADORA

_____

Prof. José Augusto Suruagy Monteiro
Centro de Informática / UFPE

_____

Prof. Rodrigo Elia Assad
Universidade Federal Rural de Pernambuco

_____

Prof. Ruy José Guerra Barretto de Queiroz
Centro de Informática / UFPE
(Orientador)

Dedico este trabalho aos meus pais, Iara e Edgar, e a minha querida esposa, Camila, que foram meus grandes incentivadores e apoiadores.

AGRADECIMENTOS

Agradecimentos especiais devem ser dedicados ao meu bom Deus pelo amor incondicional. À minha querida esposa pelo apoio e incentivo constante. Aos meus queridos familiares pelo apoio e incentivo. Aos meus grandes amigos pelo companheirismo de sempre. Ao meu orientador pela oportunidade, paciência e atenção dedicados durante o trablho. E, aos professores e colaboradores que me ajudaram durante esta caminhada.

Também, um agradecimento especial a direção e colaboradores do Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - IFRS, por todo apoio e incetivo durante meus estudos.

Deus ...

Eu pedi Força ...
e Deus me deu Dificuldades para me fazer forte.

Eu pedi Sabedoria ...
e Deus me deu Problemas para resolver.

Eu pedi Prosperidade ...
e Deus me deu Cérebro e Músculos para trabalhar.

Eu pedi Coragem ...
e Deus me deu Perigos para superar.

Eu pedi Amor ...
e Deus me deu Pessoas com Problemas para ajudar.

Eu pedi Favores ...
e Deus me deu Oportunidades.

Eu não recebi nada do que pedi ...
mas eu recebi tudo de que precisava.

(Autor Desconhecido)

RESUMO

Vários sites coletam informações sobre os usuários/dispositivos, quando são acessados ou mesmo durante a navegação, muitas vezes sem o seu consentimento. Dentre as várias técnicas existentes, a que está em foco no trabalho utiliza o método de Canvas Fingerprinting para gerar uma identificação única do usuário/dispositivo, através da coleta de algumas informações e fazendo alguns processamentos disponibilizados através da linguagem HTML5. Desta forma, o usuário/dispositivo pode ser identificado nos próximos acessos e até mesmo informações sobre ele podem ser recuperadas se já foram coletadas anteriormente. Com isso, a pesquisa foi desenvolvida com o intuito de abordar uma forma de identificar e avisar ao usuário, que acessa um determinado site, se o mesmo está fazendo sua identificação utilizando o método de Canvas Fingerprinting. Nesta busca, o estudo identificou a existência da extensão para Firefox denominada Canvas Blocker que identifica e bloqueia a utilização do mecanismo de Canvas do HTML5. Devido à existência desta extensão, o trabalho a modificou para detectar a utilização específica do mecanismo de Canvas Fingerprinting e alertar o usuário sobre essa utilização. Para isso, foi utilizada uma metodologia de detectar Canvas Fingerprinting já presente na literatura e agregado a isso algumas sugestões para reduzir os falsos negativos e os falsos positivos encontrados durante os experimentos. Então, o algoritmo proposto foi aplicado em um conjunto de sites determinado, alguns dados foram coletados dessas visitas, posteriormente estes dados foram analisados e, por fim, apresentadas medições que comprovaram a eficiência da solução apresentada.

Palavras-chave: Canvas Fingerprinting. Device Fingerprint. Identificação do Dispositivo.

ABSTRACT

Several sites collect information about users/devices, when they are accessed or even while browsing, often without consent. Among the several techniques, one that is in focus in this work uses the Canvas Fingerprinting method to generate a unique identification of the user/device, through the collection of some information and making some processing available through the HTML5 language. In this way, the user/device can be identified in the next accesses and even information about him/it can be recovered if previously collected. Thereby, the research was developed with the intention of approaching a way to identify and warn the user, who accesses a certain site, if the site is making his identification using the method of Canvas Fingerprinting. In this search, the study identified the existence of a Firefox extension called Canvas Blocker that identifies and blocks the use of the HTML5 Canvas mechanism. Because of the existence of this extension, the work modified it to detect the specific use of Canvas Fingerprinting mechanism and to alert the user about this use. For this, a methodology already present in the literature was used to detect Canvas Fingerprinting and added some suggestions to reduce the false negatives and the false positives found during the experiments. Then, the proposed algorithm was applied in a given set of sites, some data were collected from these visits, later these data were analyzed and, finally, measurements were presented that proved the efficiency of the presented solution.

Keywords: Canvas Fingerprint. Device Fingerprint. Device Identification.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS OU SIGLAS

API	Application Programming Interface (Interface de Programação de Aplicativos)
art.	artigo
GB	Gigabyte
HTML	Hyper Text Markup Language (Linguagem de Marcação de Hipertexto)
HTML5	HTML versão 5
IPC	Inter-Process Communication (Comunicação entre Processos)
IPDL	IPC Protocolo Definition (Definição do Protocolo IPC)
JPG/JPEG	Joint Photographic Experts Group, ou jpg (um formato de imagem)
JSON	JavaScript Object Notation
JS	JavaScript
log	Registro de eventos em um sistema de computador
MD5	Message-Digest algorithm 5 (função hash criptográfica de 128 bits)
RAM	Random Access Memory (Memória de Acesso Randômico)
RegEx	Regular Expression (Expressão Regular)
SEO	Search Engine Optimization (Otimização do Mecanismo de Pesquisa)
SQL	Structured Query Language (Linguagem de Consulta Estruturada)
TI	Tecnologia da Informação
URL	Uniform Resource Locator (Localizador Uniforme de Recursos)

SUMÁRIO

1 INTRODUÇÃO

Estamos vivendo num mundo onde a palavra de ordem é vender. Uns vendem suas imagens, outros suas ideias e outros produtos. Além disso, qualquer informação, hoje em dia, corre o mundo em segundos. Então, somos bombardeados com uma vasta quantidade de informações vindas de todas as partes do mundo, inclusive anúncios de venda de algum produto.

Normalmente, em anúncios online, os anunciantes somente pagam os sites publicitários quando o cliente clica em um de seus anúncios, ou seja, o anunciante paga por “cliques” no seu anúncio. Assim, tentando refinar o número de anúncios de marketing que nos são apresentados diariamente quando acessamos a Internet, muitos sites têm coletado informações do comportamento dos usuários na Internet e assim, selecionando anúncios específicos que “teoricamente” seriam de interesse do usuário. Por consequência, anúncios considerados irrelevantes, segundo algum padrão adotado, são desconsiderados e não são mostrados para o usuário. Esse tipo de refinamento, na tentativa de encontrar um padrão comportamental do usuário online, encontrado em Borgesius (2014, p.28), é chamado de *Behavioural Targeting* (ou *Behavioural Advertising* ou *Online Profiling*) ou Segmentação Comportamental (ou Publicidade Comportamental ou Perfil Online).

Conforme o modelo tradicional, as informações coletadas que identificam o usuário e caracterizam suas preferências são armazenadas utilizando *cookies* e/ou *flash cookies*. Como explicado por Borgesius (2015, p.2), *cookies* são arquivos de texto contendo informações que identificam aquele computador/usuário e por isso são armazenados no próprio computador que se quer identificar. Assim, quando o usuário acessa um site que já coletou suas informações e estas foram armazenadas nos *cookies*, basta ler as informações contidas nos mesmos e apresentar o site (ou os anúncios do site) conforme as preferências deste usuário. Ainda, Borgesius (2015, p.2) comenta que praticamente todos os sites populares utilizam um ou mais *cookies* para rastreamento.

Com o advento da Internet das Coisas o rastreamento do usuário pode

acontecer não só em computadores, mas com qualquer dispositivo e fica melhor se ele estiver conectado à Internet. Borgesius (2015, p.2-3) fornece vários exemplos, um com câmeras que tentam reconhecer a idade e gênero da pessoa, e quadros de avisos com reconhecimento facial. O mesmo autor ainda cita que o Google prevê que em alguns anos poderá fornecer anúncios em refrigeradores ou no painel de carros, por exemplo.

Mesmo assim, são poucos os sites que avisam ao usuário antes de coletar informações do comportamento do mesmo e armazená-las nos *cookies*. Isto gera um desconforto por parte de alguns usuários pois acreditam que estão sendo "espionados" sem o seu consentimento, já que não foram consultados se autorizariam essa coleta e armazenamento de informações.

Tecnicamente falando, meios de se proteger e bloquear essa coleta de informações privadas foram desenvolvidas e são utilizadas diariamente por muitos usuários. Existem mecanismos de proteção no próprio navegador (por exemplo, desabilitar os *cookies*) ou com a utilização de *plugins* que bloqueiam ou avisam tal comportamento dos sites que estão sendo visitados pelo usuário.

Outra forma de rastreamento dos usuários é encontrada nos apontamentos de Saraiva et al. (2014, p.1), são as técnicas de *device fingerprinting* empregadas para identificar um usuário ou dispositivo através de dados coletados durante o processo de comunicação. Os usuários sabem pouco ou quase nada sobre o assunto, mas essas técnicas estão cada vez mais presentes na Web e os usuários não são informados sobre elas (SARAIVA et al., 2014, p.45). Os autores relatam que técnicas mais recentes de *device fingerprinting* podem burlar *plugins* e extensões para navegadores destinados a bloquear este tipo de rastreamento. Por isso, os autores (SARAIVA et al., 2014, p.46) sugerem que mais pesquisas sejam desenvolvidas na área, inicialmente resolvendo problemas específicos.

Com a chegada do HTML5, novas funcionalidades foram disponibilizadas para facilitar a vida dos programadores web. Dessas funcionalidades, uma em especial deve ser mencionada, a que possibilita fazer "desenhos" durante a geração da página web, em cima de um canvas (uma lona, uma espécie de tela de desenho). Com isso, surge a possibilidade de criar um desenho e em cima desse gerar um *device fingerprinting*, utilizando esse desenho. Esta técnica passou a ser conhecida como "*canvas fingerprinting*" e é uma forma de coletar informações sobre um computador remoto com o objetivo de identificar de forma única o dispositivo ou o usuário sem a utilização de *cookies*. Aparentemente, por se tratar de uma

ferramente recente, pouco se sabe sobre ela, nem se realmente gera uma identificação única de cada dispositivo/usuário, nem se existe uma forma de se proteger contra essa identificação que acontece sem o consentimento do usuário.

1.1 Objetivos

Além de buscar formas de facilitar a implementação de novas aplicações e funcionalidades (como no HTML5), outro assunto que está sempre em vista é a segurança da informação. Principalmente na atualidade, quando existem tantas formas de compartilhar e acessar a mais vasta gama de informações. Por isso, tanto no ambiente do usuário como no corporativo a segurança da informação sempre está entre as prioridades a serem garantidas.

Em linhas gerais, a segurança da informação se apoia em três pontos fundamentais: confidencialidade, integridade e disponibilidade da informação. Para este trabalho iremos focar na confidencialidade do usuário durante a navegação na Internet.

Já no que concerne à segurança da informação em organizações, conforme os apontamentos de Whitman e Mattord (2012, p.41) a segurança da informação engloba quatro pontos fundamentais:

1. Proteger a habilidade da organização funcionar;
2. Assegurar a segurança das operações nos sistemas de TI da organização;
3. Proteger os dados coletados e utilizados; e
4. Garantir a segurança do patrimônio da organização.

Desta forma, o trabalho busca garantir o anonimato do usuário durante a navegação na Internet (se ele assim desejar) e, consequentemente, a confidencialidade e proteção de seus dados, para que estes não sejam acessados ou processados sem o seu consentimento.

Tudo isso quando a forma de identificação do usuário/dispositivo utilizar o mecanismo de C*anvas Fingerprint*, que é o foco do trabalho. E, para garantir o

anonimato de forma mais específica, tem-se como objetivo aplicar uma técnica de detecção da utilização do mecanismo de *Canvas Fingerprinting* encontrada na literatura.

Para isso, disponibilizaremos uma extensão compatível com o navegador Firefox. Utilizaremos a extensão Canvas Blocker na implementação, faremos uma modificação na mesma para detectar e avisar ao usuário quando a página que está sendo acessada coletou dados e fez sua identificação utilizando o mecanismo de Canvas Fingerprinting.

Ainda, como meio de teste, será feito um experimento para aplicar a solução adotada nos primeiros registros do Ranque Alexa (lista já adotada no trabalho de Acar et al. (2014)), relatando os dados obtidos. E, durante a identificação das páginas que utilizam o mecanismo de *Canvas Fingerprinting* nessa lista de sites, buscam diferenciar as detecções positivas dos falsos positivos, ou seja, refinar as detecções.

Além disso, para garantir a eficiência da solução adotada, será aplicado um método de medição da solução adotada. Com esta, realizar-se-á uma comparação com outras soluções pesquisadas, acompanhada das devidas críticas.

Desta forma, o trabalho está estruturado como segue. No Capítulo 2, uma breve descrição da base legal que busca proteger a privacidade do usuário, são mostrados alguns trabalhos relevantes relacionados a Fingerprinting, a Canvas Fingerprinting e contramedidas sobre o tema, inclusive com explanação sobre a extensão Canvas Blocker. Depois, no Capítulo 3, é relatado o funcionamento e as modificações feitas na implementação da extensão Canvas Blocker, as principais mudanças propostas para o mecanismos de bloqueio, formas de reduzir falsos negativos e reduzir falsos positivos. Na sequência, no Capítulo 4, são explanadas as definições feitas durante a implementação, explicações de como foram realizadas as visitas às páginas web para a coleta dos dados, como foi feita a organização destes e sua análise, chegando nos resultados que foram encontrados no decorrer desta pesquisa. Por fim, no Capítulo 5 é apresentada a conclusão dos estudos aqui apresentados, discutidos os trabalhos futuros e sugestões de melhorias.

2 FINGERPRINTING E CANVAS FINGERPRINTING

Neste capítulo são apresentadas algumas bases legais internacionais e nacionais que obrigam o site a avisar ao visitante quando são coletados e processadas informações desses usuários. Em sequência são apresentados formas de identificar o usuário utilizando Fingerprinting e Canvas Fingerprinting. Em seguida são apresentadas algumas contramedidas e, por fim, é feita uma explanação sobre a extensão Canvas Blocker.

2.1 Base Legal

Falando sobre o processamento de informações privadas, além das discussões morais, existe toda uma discussão social e legal por trás desta coleta e processamento das informações privadas. Borgesius (2015), levando em consideração as leis que regem a União Europeia, faz toda uma reflexão a respeito, apontando fatos que defendem e contrapõem o processamento dessas informações privadas, principalmente quando obtidas sem o consentimento do usuário. O autor (2015, p.8) argumenta que qualquer processamento de dados privados deve ter autorização do proprietário desses dados.

Ainda, conforme Artigo 7(f) do *Data Protecting Directive* (uma normativa da União Europeia) citado por Borgesius (2015, p.8), se o provimento do serviço, pelo web site, depender desse processamento de dados a autorização pode ser dada ao aceitar os “Termos e Condições” (*Terms and Conditions*) ou através de uma outra base legal. Em caso contrário, apenas um consentimento explícito do usuário provê um fundamento legal para realizar o processamento desses dados. E o autor diz mais (2015, p.10), resumindo o Artigo 5(3) do *e-Privacy Directive* (uma das

normativas da União Europeia sobre dados pessoais e privacidade nas comunicações eletrônicas) que complementa o artigo anterior (2015, p.13), afirma que qualquer um que armazena dados ou acesse informações em qualquer dispositivo do usuário deve solicitar seu consentimento. E conclui (2015, p.14) que o consentimento de armazenar *cookie* com dados pessoais não é o mesmo consentimento para processar dados pessoais, eles devem ser explícitos separadamente e consentidos pelo usuário.

Outros autores, Arnold, Hillebrand e Waldburger (2015, p.12-18), também afirmam que, para os países membros da União Europeia, o direito de proteção dos dados pessoais é direito fundamental e implementado na normativa da *European Data Protection Directive* e complementada na *e-Privacy Directive*. Da mesma forma que o outro autor, concluem que em alguns casos não precisa de consentimento, como por exemplo um *cookie* utilizado para comunicação de dados. Mas quando se coleta e armazena dados pessoais essa autorização deve ser expressa e especificamente dada pelo usuário em questão.

Uma questão interessante que Arnold, Hillebrand e Waldburger (2015, p.20-32) levantam é o problema da assinatura sem leitura ou do clique sem leitura, ou seja, muitas das mensagens (se não todas) que são mostradas para os usuários, quando estão navegando na Internet ou instalando um aplicativo, não são lidas por eles. Isso acontece aceitando os “Termos e Condições” do serviço ou mesmo dando “OK” em uma mensagem que aparece na tela, sem ler. Desta forma, impactando diretamente sobre quem pode acessar, analisar e manipular seus dados pessoais. Um dos motivos para não ler os Termos e Condições, levantada pelos autores (ARNOLD; HILLEBRAND; WALDBURGER, 2015, p.25-30), é a sua extensão e a dificuldade de compreensão, mesmo por estudantes de Direito.

Mas Arnold, Hillebrand e Waldburger (2015, p.32-38) afirmam que mesmo não concordando com os termos de acesso a seus dados pessoais, o consumidor acaba aceitando porque quer utilizar o serviço/aplicação ou porque tem amigos e conhecidos que já utilizam e quer compartilhar informações com eles.

Como pode ser visto, na União Europeia a discussão sobre o tema está bastante recente e vários dos países membros já utilizam as leis que garantem a privacidade e a necessidade de um consentimento expresso do usuário para coletar, armazenar e processar dados privados do mesmo. Servindo este consentimento tanto para o conhecimento do usuário como uma base legal para a ação de processamento dos dados.

Também no Brasil o legislador já vem se preocupando com a questão já faz algum tempo. A Constituição Federal prevê em seu artigo 5º, inciso XII:

> Art. 5º (…)
> XII - é inviolável o sigilo da correspondência e das comunicações telegráficas, de dados e das comunicações telefônicas, salvo, no último caso, por ordem judicial, nas hipóteses e na forma que a lei estabelecer para fins de investigação criminal ou instrução processual penal;

Mais recentemente, foi aprovado o "Marco Civil da Internet" (Lei nº 12.965 de 23 de abril de 2014) em que um de seus princípios é a proteção à privacidade. E, mais especificamente sobre a coleta e processamento de dados privados temos o Art. 7º, incisos VII, VIII e IX que dizem:

> Art. 7º O acesso à internet é essencial ao exercício da cidadania, e ao usuário são assegurados os seguintes direitos:
> (...)
> VII - não fornecimento a terceiros de seus dados pessoais, inclusive registros de conexão, e de acesso a aplicações de internet, salvo mediante consentimento livre, expresso e informado ou nas hipóteses previstas em lei;
> VIII - informações claras e completas sobre coleta, uso, armazenamento, tratamento e proteção de seus dados pessoais, que somente poderão ser utilizados para finalidades que:
> a) justifiquem sua coleta;
> b) não sejam vedadas pela legislação; e
> c) estejam especificadas nos contratos de prestação de serviços ou em termos de uso de aplicações de internet;
> IX - consentimento expresso sobre coleta, uso, armazenamento e tratamento de dados pessoais, que deverá ocorrer de forma destacada das demais cláusulas contratuais;
> (…)

Ou seja, como vem acontecendo em outros países, o Brasil também já está normatizando a coleta, armazenamento e processamento de informações feita pela grande maioria dos sites da Internet. Prevendo que essa coleta e processamento deve ser informada e consentida de forma expressa pelo usuário para ter validade legal.

Assim, como vem se concretizando em diversos países, inclusive no Brasil, a aprovação para coleta, armazenamento e processamento de dados privados tem que ser explicitamente fornecida pelo usuário, o que não ocorre na grande maioria das vezes. Todavia, em um grande número de páginas a navegação completa somente é possível mediante a coleta de dados pessoais. Adicionalmente a isso, com o mecanismo do "*Canvas Fingerprinting*" essa coleta de dados pessoal e identificação do dispositivo se tornou ainda mais fácil, sendo feita sem o consentimento ou conhecimento do usuário. E, como já temos uma forma legal de exigir esse consentimento por parte dos sites, só falta um mecanismo que comprove o rastreamento que acontece sem o nosso consentimento.

2.2 Formas de Identificar o Usuário

Existem várias pesquisas e técnicas empregadas na tentativa de identificar de forma única usuários Web. Embora muitas das técnicas de *Fingerprinting* baseadas na Web sejam aplicações legítimas, outras são utilizadas para obter e explorar informações pessoais de usuários, muitas vezes, sem o seu consentimento (KHADEMI; ZULKERNINE; WELDEMARIAM, 2015).

De forma simplificada, para gerar um *fingerprint* (da forma tradicional) como nos apresenta Eckersley (2010), são coletadas informações do dispositivo (como resolução de tela, informações sobre o navegador, sobre *cookie*, sobre flash, sobre *plugins*, etc) e gerada uma *string* com todas essas informações. Em cima dessa *string* é gerado um *hash* (número) e esse número seria a identificação (*fingerprint*) do dispositivo/usuário.

Com o objetivo de gerar um *fingerprinting* vários estudos recentes estão sendo desenvolvidos para melhorar a geração desse identificador, tornando-o o mais único possível (entropia elevada). Recentemente, Khademi, Zulkernine e Weldemariam (2015) desenvolveram uma pesquisa que abrange as quatro técnicas de *Fingerprinting* baseadas em Web utilizadas nos sites atuais: *JavaScript object fingerprinting*, *JavaScript-based font detection*, *canvas fingerprinting* e *Flash-based fingerprinting*. Com este estudo, desenvolveram uma ferramenta chamada Fybrid

onde combinam os quatro métodos estudados para criar um *fingerprint* da instância do navegador utilizado no acesso. Fifield e Egelman (2015) fazem um estudo sobre a métrica das fontes de letras, definem uma técnica de *Fingerprinting* do navegador baseada nas características dessas fontes de letras utilizadas pelo navegador, para gerar um *fingerprint*, ou em conjunto com outras técnicas, para ajudar a individualizar um usuário.

Tendo em vista especificamente o *Canvas Fingerprinting*, existem duas formas de obter o identificador (*fingerprint*). Na primeira, como relatada por Acar et al. (2014), é gerado um identificador utilizando a própria imagem gerada no *canvas*. Já na segunda forma, observada na biblioteca FingerprintJS2[1] estudada, é gerada uma *string* composta por informações coletadas do navegador em conjunto com uma *string* no formato *dataURL* (uma representação codificada na base 64 dos dados de *pixel* binário) obtida a partir da imagem de c*anvas* gerada dinamicamente pela biblioteca FingerprintJS2. Então, nesta segunda forma, o *fingerprint* é gerado em cima dessa *string* mencionada.

A biblioteca FingerprintJS2 é a versão 2 da biblioteca FingerprintJS[2], ambas desenvolvidas pelo mesmo autor, Valentin Vasilyev (VALVE). Tanto a versão 1 como a versão 2 da biblioteca são utilizadas para gerar um *fingerprint* para fazer a identificação do usuário que está visitando o site. Contudo, ela apenas gera o identificador, não avisa ao usuário da sua execução.

Um dos primeiros autores a falar especificamente sobre *Canvas Fingerprinting* foi Mowery e Shacham (2012), em seu artigo "*Pixel Perfect: Fingerprinting Canvas in HTML5*", onde falam sobre a utilização das especificações da suíte do HTML5 para gerar um *fingerprint*.

Estas novas especificações do HTML5 disponibilizaram uma área de desenho programático (<canvas>), gráfico tridimensional (WebGL), estrutura de armazenamento de dados no lado do cliente, serviço de geolocalização, possibilidade de manipular o histórico e o cache do navegador, reprodução de áudio e vídeo, dentre outras funcionalidades.

Mowery e Shacham (2012) demonstram que o comportamento da renderização de <*canvas*> *text* (texto em camadas) e utilizando WebGL nos navegadores atuais poderia ser utilizado como um novo sistema de *Fingerprinting* (impressão digital), ou seja, uma nova forma para identificar dispositivos/usuários

1 Site da biblioteca FingerprintJS2 em https://github.com/Valve/fingerprintjs2
2 Site da biblioteca FingerprintJS em https://github.com/Valve/fingerprintjs

que navegam na Internet.

Ainda, segundo os autores (MOWERY, SHACHAM, 2012), o novo sistema de identificação pode ser considerado:

- Consistente: ter base sólida, aplicando o mesmo algoritmo no mesmo ambiente resulta no mesmo identificador;
- Entropia elevada: implica em maior desordem, na grande maioria das vezes, gera padrões diferentes em dispositivos diferentes;
- Ortogonal a outros *fingerprints*: mais simples, precisa utilizar menos construções que outros *fingerprints*;
- Transparente para o usuário: não é perceptível ao usuário quando está gerando o identificador; e
- Prontamente obtido: em pouquíssimo tempo gera o *fingerprint*.

Mowery e Shacham (2012) não sabem se será possível identificar toda a entropia da Internet, mas os testes foram promissores. Os autores não concordam em eliminar essa forma de *Fingerprint*, pois a performance do navegador não sofreu alteração significativa e evitaria novas perguntas de autorização ao usuário para execução de funcionalidades básicas. Talvez chegou o momento de reconhecer que as impressões digitais (*fingerprints*) sejam inevitáveis na web moderna.

Outro artigo que deve ser mencionado sobre o assunto é o "*The Web Never Forgets: Persistent Tracking Mechanisms in the wild*" (ACAR et al., 2014). Neste, os autores fazem uma pesquisa sobre *canvas fingerprinting*, *evercookies* e *cookie syncing* em conjunto com *evercookies*. Mencionam que, dentre os 100.000 web sites pesquisados, 5,5% empregam *canvas fingerprinting*. Sobre este assunto, aparentemente não existe uma forma de bloquear automaticamente *Canvas Fingerprinting* sem falsos positivos, o que bloquearia também funcionalidades legítimas. Mesmo uma correção parcial requereria um *patch* de código-fonte do navegador.

Quando um *canvas fingerprint* é gerado, segundo Acar et al. (2014), o resultado da renderização do texto escolhido pode ser afetado de diferentes formas dependendo do sistema operacional, da biblioteca de fontes de letras, da placa gráfica, do drive gráfico e do navegador. Assim, para aumentar a diversidade de resultados é bom utilizar uma frase com o máximo de letras diferentes possível. Por isso, uma das frases utilizadas nos experimentos foi o pangrama (ou pantograma)

"*How quickly daft jumping zebras vex*".

O fluxo para gerar um *canvas fingerprint*, quando um usuário visita uma página web, segundo Acar et al. (2014), passa pelos seguintes passos:

i. Primeiramente o script de *fingerprinting* desenha o texto com a fonte e o tamanho escolhidos, e adiciona cores de fundo;
ii. Em seguida o script chama o método `ToDataURL` da API de Canvas para converter os dados de *pixel* do *canvas* no formato `dataURL`. Basicamente, este formato é uma representação codificada na base 64 dos dados de *pixel* binário; e
iii. Por fim, o script gera um *hash* dos dados de *pixel* codificados. Este *hash* serve como *fingerprint*. Mas pode ainda ser combinado com outras propriedades do navegador para uma maior entropia, como a lista de *plugins*, a lista de fontes de letra, ou com a *string* de usuário.

Na pesquisa realizada por Acar et al. (2014) foram analisados os primeiros 100.000 links do ranque da Alexa. Eles conseguiram identificar as formas de detectar *Canvas Fingerprinting*, desenvolveram um rastreador baseado num navegador modificado e executaram rastreamentos exploratórios. Eles desenvolveram uma detecção quase totalmente automatizada de *Canvas Fingerprinting*. E, também apontam os seguintes procedimentos para eliminar falsos positivos e melhorar a detecção de Canvas Fingerprinting:

- Devem estar presentes ambas as chamadas para os métodos `ToDataURL` e `fillText` (ou `strokeText`), e ambas virem da mesma URL: para se certificar que o *script* gerou a imagem e depois vai ler a imagem, descartando aqueles que apenas vão gerar ou ler uma imagem;
- A imagem de *canvas* lida pelo *script* deve conter mais de uma cor e seu tamanho agregado deve ser maior que 16x16 *pixels*: isso para descartar *scripts* que tentam gerar um *fingerprint* com uma imagem muito pequena, pois não são eficientes e geram identificadores com baixa entropia; e
- A imagem não deve ser requisitada em um formato compactado, como JPEG por exemplo: para descartar imagens que utilizam compressão com perdas, pois a imagem retornada pode perder as diferenças sutis que são essenciais

para gerar um identificador (*fingerprint*) de entropia elevada.

Assim, todas essas propriedades levantadas sobre o mecanismo de *Canvas Fingerprinting* podem ser utilizadas para desenvolver uma ferramenta para detectar esse a utilização desse método, se proteger e garantir o anonimato do usuário, se assim desejar.

2.3 Algumas Contramedidas Existentes

Basicamente, as pesquisas na literatura nos levam a perceber que existem duas formas para prevenir os indesejados *fingerprints*. Uma consiste em bloquear o rastreador, ou seja, bloquear o mecanismo que gera o identificador e não permitir que o mesmo gere um *fingerprint*. Outra forma é modificar as variáveis que ele utiliza para gerar um *fingerprint*, gerando assim um identificador diferente do esperado. Ambas as formas têm pontos positivos e negativos, são mencionadas por Torres, Jonker e Mauw (2015) onde completam dizendo que nenhuma é totalmente eficiente como contramedida para este tipo de rastreamento.

Pensando em navegação privada, uma das primeiras opções que nos vem à mente é o navegador Tor (*The Onion Router*). No site oficial (*Site Tor Browser*) encontramos que este programa objetiva melhorar a privacidade e a segurança do usuário. Para isso, como uma de suas principais características, em vez de fazer uma conexão direta ao site desejado, utiliza um túnel (criptografado) através de servidores roteadores (chamados de *relays*) para rotear os dados dentro da rede Tor. O navegador muda a rota de tempos em tempos e para endereços diferentes, sempre criptografando a informação dentro da rede Tor (acessos finais, externos à rede, não são criptografados). Com essa técnica, o navegador Tor não resolve todos os problemas de privacidade, mas garante a proteção no transporte de dados, ou seja, de certa forma garante que sua localização não seja identificada e que os dados sejam transportados de forma criptografada (dentro da rede Tor). No entanto, para garantir que informações de identidade não sejam conseguidas pelos sites visitados, outros métodos devem ser usados ou agregados a este.

Outra contramedida é o *plugin* FireGloves[3]. A ideia desta extensão surgiu do estudo sobre *fingerprint* feito por Boda et al. (2012), que discutiu uma técnica de *Fingerprint* independente de navegador onde, inicialmente, uma parte do endereço IP, a disponibilidade de um conjunto específico de fontes, o fuso horário e a resolução de tela seriam suficientes para gerar um *fingerprint* que identificasse a maioria dos usuários. O FireGloves era uma extensão criada por pesquisadores da Universidade de Tecnologia e Economia de Budapeste para demonstrar a possibilidade de se defender contra sistemas de *Fingerprint*. Sua última versão foi disponibilizada em 2012, mas foi uma das primeiras, senão a primeira, extensão lançada com esse propósito.

O grupo de Roesner, Kohno e Wetherall (2012) também desenvolveu uma extensão para o Firefox chamada de ShareMeNot. O objetivo era proteger o mapeamento do comportamento do usuário e não ser rastreado por programas de rastreamento de terceiros, inseridos na página em que se está navegando. Ele não bloqueia completamente, deixa a escolha para o usuário. Por exemplo, se tem um botão de terceiro na pátina que rastreia a atividade do usuário, o ShareMeNot impede o rastreamento automático, mas se o usuário clicar no botão ele carrega os *cookies* associados e permite o rastreamento. Assim, o ShareMeNot se torna eficiente por fazer esse meio termo de bloquear código de terceiro incluído na página, mas liberar se o usuário interagir com ele.

Nessa linha, uma extensão para o Firefox chamada Priv3[4] desenvolvida em 2011 por Dhawan, Kreibich e Weaver (2011) faz algo parecido ao ShareMeNot. A extensão Priv3 bloqueia *cookies* de terceiros (das redes sociais Facebook, Twitter, Google +1 e LinkedIn) durante a navegação na Web, mas se o usuário interagir com eles seu conteúdo é liberado.

Com Nikiforakis, Joosen e Livshits (2014), através de uma técnica de geração randômica de alguns parâmetros do navegador (que são utilizados como informação para gerar o *fingerprint*), foi proposta uma modificação no código fonte do navegador Chromium, no seu modo de navegação privada, denominado PriVaricator. Segundo o autor, a técnica não está em prevenir o *fingerprint* mas em não deixar associar o *fingerprint* (gerado em vários sites) a um usuário. Assim, o PriVaricator foi incorporado ao navegador para fazer com que seja gerado um *fingerprint* diferente em cada visita e, desta forma, dificilmente esse identificador

3 Site do FireGloves em https://fingerprint.pet-portal.eu/?menu=6, ainda, maiores detalhes podem ser encontrados no site https://pet-portal.eu/blog/read/533/?set_language=eng. Código fonte em

4 Extensão Priv3 disponível em http://priv3.icsi.berkeley.edu/

possa ser associado com outro *fingerprint*, resultante de outra visita ao site, o que dificultaria o rastreamento.

Algo parecido ao que é proposto no ShareMeNot e no Priv3 é encontrado no FP-Block, pois ambos buscam bloquear o rastreamento de terceiros. Para isso, Torres, Jonker e Mauw (2015) realizam uma pesquisa e a implementação de um *plugin* chamado de FP-Block (FingerPrint-Block). Esta extensão busca garantir que qualquer código de terceiro, incorporado ao site visitado, que tente gerar uma identificação (*fingerprint*) tenha cada vez uma impressão digital diferente, pois leva em consideração o site (domínio) no qual ele é incorporado. Em resumo, gera um identificador (*fingerprint*) diferente em páginas web diferentes para *fingerprints* de terceiros. Segundo o autor, este aplicativo não interfere no rastreamento do site principal, apenas no rastreamento de terceiros. Isso garantiria, por exemplo, que botões de mídia social de terceiros (como do Facebook), incorporados a um site, não consigam rastrear o usuário em diferentes sites. Desta forma, permitiria a geração do *fingerprint*, mas geraria um diferente para cada site (domínio) que estivesse sendo visitado. A limitação está em focar apenas em duas camadas de comunicação: JavaScript e HTTP. Assim, segundo o autor, o FP-Block é capaz de bloquear os códigos de terceiros do Facebook, Twitter, Google Plus, LinkedIn, Tumblr e Pinterest.

Além disso, foram identificadas duas extensões que têm como foco interferir no mecanismo de *Canvas Fingerprinting* e consequentemente na geração do *fingerprint*. A primeira, é a extensão Canvas Blocker que será mostrada na seção 2.4 a seguir. A outra, é uma extensão chamada de Canvas Defender[5] que, segundo a documentação, não bloqueia a geração do *canvas fingerprint*, mas gera um ruído (*noise*) que esconderia seu c*anvas fingerprint* real. Este ruído é uma espécie de *hash* (uma sequência numérica aleatória) que pode ser gerada ao pressionarmos o botão presente na extensão (ou seja, podemos trocar o *hash* quando quisermos) ou um novo *hash* é gerado quando abrimos o navegador Firefox pela primeira vez. Contudo, não é nosso objetivo verificar o funcionamento da extensão Canvas Defender. O que constatamos é que ela não avisa que o site visitado está utilizando o mecanismo de *Canvas Fingerprinting*, mas, teoricamente, interfere na geração do *fingerprint* em questão.

Assim, as contramedidas apresentadas são eficientes, teoricamente, no seu

5 Extensão Canvas Defender disponível em https://addons.mozilla.org/en-US/firefox/addon/no-canvas-fingerprinting/ para Firefox, mas também existe uma versão desta para o navegador Chrome.

conjunto de atuação para tratar a geração de alguns *fingerprints* específicos, mas não avisam ao usuário da utilização do mecanismos de *Canvas Fingerprinting*. Contudo, por exemplo, na biblioteca FingerprintJS2 é utilizado o *Canvas Fingerprinting* em conjunto com a coleta de algumas informações do navegador. Desta forma, quando um site gerar um *fingerprint* utilizando a biblioteca FingerprintJS2, a técnica abordada por Nikiforakis, Joosen e Livshits (2014), por exemplo, pode modificar o *fingerprint* e resultar num identificador diferente do esperado, auxiliando no anonimato do usuário.

2.4 Extensão Canvas Blocker

O programa Canvas Blocker nada mais é que uma extensão para o Firefox que bloqueia algumas funcionalidades da API de Canvas do HTML5. E, teoricamente, bloqueando essas atividades ele bloquearia também as tentativas de utilização do mecanismo de *Canvas Fingerprinting*.

Segundo o Site do Canvas Blocker, este programa avisa ao usuário quando a API de Canvas for utilizada para prevenir o *Fingerprinting* do mesmo. Os usuários podem bloquear a utilização da API de Canvas completamente ou parcialmente por um web site, ou ainda podem gerar um canvas "*fake*" (falso), que é um *fingerprint* parecido com o solicitado mas com algumas modificações na imagem que são feitas dinamicamente pela extensão.

Esta extensão para o Firefox tem o código fonte disponível em <*https://github.com/kkapsner/CanvasBlocker*>. Este foi o código fonte utilizado como base e sobre o qual as sugestões de melhoria foram implementadas para aprimorar as detecções de Canvas Fingerprinting.

Genericamente falando, no Site do Canvas Blocker, encontramos que as diferentes formas de bloqueio que o programa permite são:

- **bloqueio de leitura da API (*block readout API*):** sites que não estão na lista branca (*white list*) ou na lista negra (*black list*) podem utilizar a API de *Canvas*, mas a API de leitura não retorna nenhum valor;

- **API de leitura falsa (*fake readout API*):** é a opção padrão. Sites que não estão na lista branca (*white list*) ou na lista negra (*black list*) podem utilizar a API de *Canvas*, mas a API de leitura retorna um valor randômico cada vez que é chamada;

- **solicitar autorização para API de leitura (*ask for readout API permission*):** sites que não estão na lista branca (*white list*) ou na lista negra (*black list*) podem utilizar a API de *Canvas*, mas é preciso solicitar ao usuário se o web site pode usar a API de leitura cada vez que ela for chamada;

- **bloquear tudo (*block everything*):** ignora as listas e bloqueia a API de *Canvas* em todos os web sites;

- **permitir apenas a lista branca (*allow only white list*):** apenas web sites que estiverem na lista branca podem usar a API de *Canvas*;

- **solicitar permissão (*ask for permission*):** se não estiver listado nas listas, pergunta ao usuário se o web site pode usar a API de *Canvas* em cada vez que for chamada;

- **bloquear apenas a lista negra (*block only black list*):** bloqueia a API de Canvas apenas dos sites que estão na lista negra; e

- **permitir tudo (*allow everything*):** ignora as listas e permite a utilização da API de *Canvas* em todos os web sites.

Então, podemos constatar que quando a extensão Canvas Blocker bloqueia a API de Canvas do HTML5, ela bloqueia também toda e qualquer atividade que utiliza a API de *Canvas*. Desta forma, teoricamente, bloqueia não apenas a utilização do mecanismos de *Canvas Fingerprinting*, mas bloqueia também muitas outras atividades que utilizam esta API para aplicações legítimas e não para identificar o usuário. Assim, o objetivo é refinar este bloqueio para minimizar os falsos positivos, identificando a utilização do mecanismos de Canvas Fingerprinting, bloqueando apenas os casos em que o site estiver fazendo a identificação do usuário/dispositivo.

3 EXTENSÃO PARA DETECÇÃO DE CANVAS FINGERPRINTING

Lembrando, o objetivos deste trabalho é disponibilizar uma extensão para o navegador Firefox que detecte e avise ao usuário quando a página que estiver sendo acessada coletar dados e fizer sua identificação utilizando o mecanismo de Canvas Fingerprinting.

Assim, neste capítulo iremos apresentar a pesquisa em si. Explicar detalhes do funcionamento do programa Canvas Blocker e o funcionamento das modificações propostas. Serão ressaltadas as principais mudanças sugeridas, formas de reduzir falsos negativos e reduzir falsos positivos conforme experimentos feitos durante a pesquisa.

3.1 Funcionamento e Modificações da Implementação

Partindo da concepção de que a extensão utiliza o conceito de múltiplos processos presente no Firefox, pode-se observar que ela também é composta por vários arquivos JS (JavaScript), cada um com uma parte do código fonte da extensão e com suas funcionalidades específicas. Assim, para um melhor entendimento, a relação entre os arquivos JS da extensão Canvas Blocker e parte do funcionamento dos múltiplos processos no Firefox podem ser vistos na imagem da Figura 3.1 abaixo.

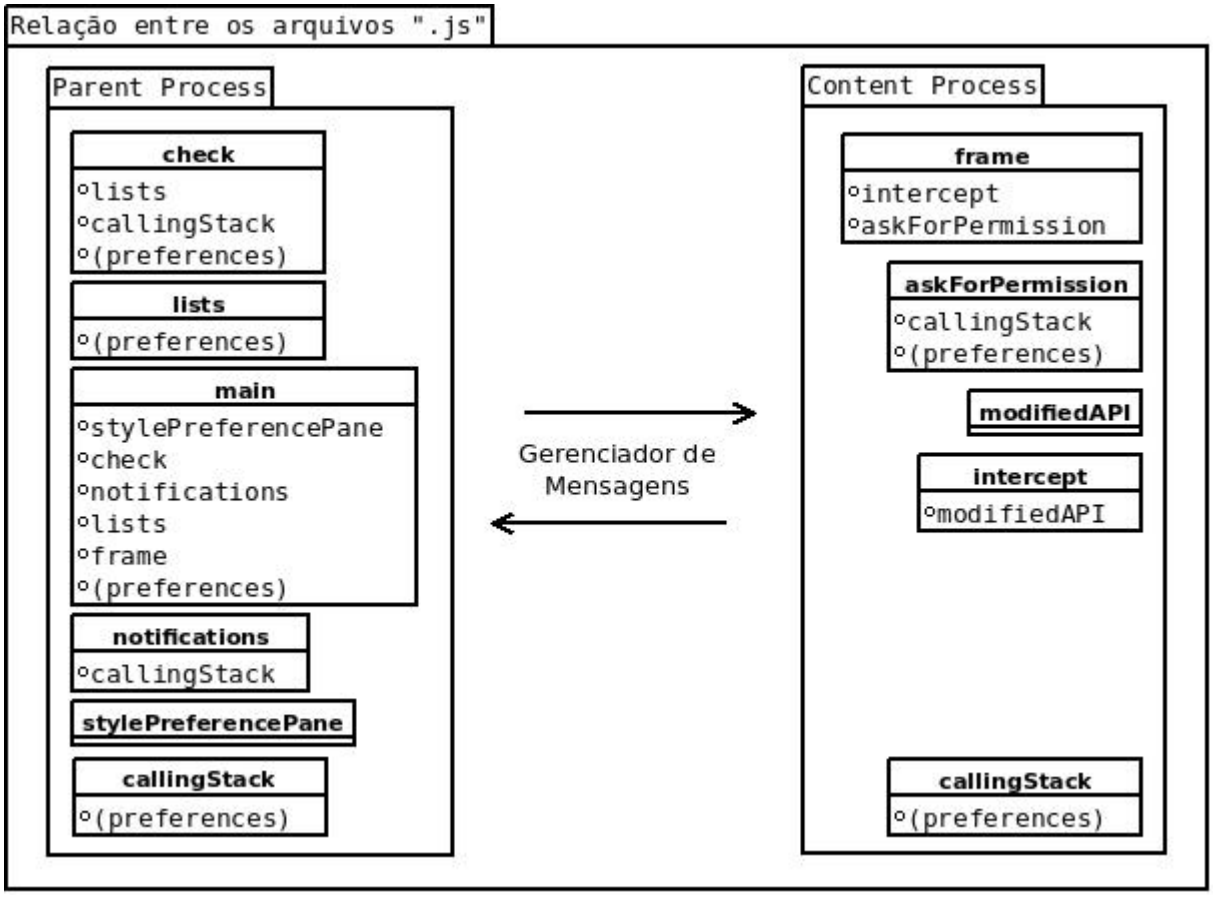

Figura 3.1 - Relação entre os arquivos JS da extensão *Canvas Blocker* e o esquema de múltiplos processos no Firefox.

Na Figura 3.1 observamos os dois processos que o Firefox utiliza, o *Parent Process* e o *Content Process*. No *Parent Process* os fontes JS da extensão que são visíveis são os dos arquivos `check`, `lists`, `main`, `notifications`, `stylePreferencePane` e `callingStack`. No *Content Process* são visíveis os arquivos `frame`, `askForPermission`, `modifiedAPI`, `intercept` e `callingStack`.

Nessa Figura 3.1 também podemos observar os *includes* de cada um dos arquivos. Por exemplo, o arquivo `frame`(.js) tem *includes* para o `intercept`(.js) e `askForPermission`(.js). Em outro exemplo, no `modifiedAPI`(.js) podemos ver que ele não tem *includes*.

Especificamente falando na implementação do Canvas Blocker, ela habilita os múltiplos processos quando define, no arquivo "`package.json`" (que não aparece na Figura 3.1, mas faz parte dos fontes da extensão), o parâmetro "`multiprocess: true`". Por isso, na implementação, utiliza-se o gerenciador de

mensagens para fazer a comunicação entre o Processo Pai (também chamado de *Chrome Process*) e o Processo Filho (*Content Process*), e vice-versa. Isso ocorre pois não se pode acessar diretamente nem o conteúdo web pelo *Parent Process* e nem o conteúdo do processo pai pelo *Child Process*, quando utiliza-se esse modo de múltiplos processos.

No arquivo "`package.json`" também são fixadas as opções padrão da extensão, utilizando a sintaxe do JSON (acrônimo para *JavaScript Object Notation*). Assim, as opções padrão estão definidas para utilizar o modo de bloqueio "`fakeReadout`" e para mostrar a notificação da utilização do mecanismo de Canvas, se assim for detectada.

Foi essa configuração padrão a adotada em todo o experimento. Ou seja, não testamos as várias combinações que os parâmetros da configuração da extensão permitiam, mas utilizamos o modo de bloqueio "`fakeReadout`" e optamos por mostrar a notificação de detecção de Canvas sempre que esta for constatada.

Ainda, voltando à Figura 3.1, o arquivo "`main.js`" é o principal carregado pelo *Parent Process* e o arquivo "`frame.js`" é o principal carregado pelo *Content Process*. Nestes arquivos são setados os eventos que devem ser esperados (no *Content Process*, em "`frame.js`") e as mensagens que devem ser esperadas (no *Parent Process*, em "`main.js`"). Com isso, através do gerenciador de mensagens global que o Firefox utiliza, resumidamente, usa-se o comando "`addMessageListener`" para registrar um ouvinte da mensagem especificada, o comando "`sendSyncMessage`" para enviar uma mensagem e um objeto de forma síncrona (para esperar a execução de todos os ouvintes), o comando "`sendAsyncMessage`" para enviar uma mensagem de forma assíncrona e o comando "`addEventListener`" para adicionar um monitor (ouvinte) para um evento específico que executa a função passada quando o evento ocorrer.

Desta forma, para toda a engrenagem de bloqueio funcionar, no arquivo "`frame.js`", com o comando "`addEventListener('DOMWindowCreated', interceptWrapper)`" é adicionado um monitor que controla a ocorrência do evento "`DOMWindowCreated`". Este evento acontece quando o objeto *Window* está sendo criado, logo no início do carregamento de cada página web. Quando este evento ocorre ele chama a função "`interceptWrapper`" que por sua vez chama a função "`intercept`", esta última definida no arquivo "`intercept.js`".

Para exemplificar, abaixo é mostrada uma parte do arquivo “`main.js`”, principal arquivo carregado pelo *Parent Process*:

```
...
11      const {check} = require("./check.js");
12      const {notify} = require("./notifications");
...
35      const {Cc, Ci} = require("chrome");
36      var globalMM = Cc["@mozilla.org/globalmessagemanager;1"].getService(Ci.nsIMessageListenerManager);
37      var frameURL = require("sdk/self").data.url("frame.js?" + Math.random());
38      globalMM.loadFrameScript(frameURL, true);
39
40      var listeners = [];
41      function addMessageListener(name, func){
42          listeners.push({name, func});
43          globalMM.addMessageListener(name, func);
44      }
...
56  //» EJr:
57      addMessageListener("GotLoadEvent", function (msg) {
58          let isCF = msg.data.isCF;
59          var browser = msg.target;
60          if (isCF) {
61              notify(msg.data, {lists, _, notificationPref, browser});
62          }
63      });
...
69      addMessageListener("canvasBlocker-check", function(ev){
70          var status = check(ev.data);
71          return status;
72      });
73
74      addMessageListener("canvasBlocker-notify", function(ev){
75          var browser = ev.target;
76          notify(ev.data, {lists, _, notificationPref, browser});
77      });
...
```

Figura 3.2 - Trecho de código do arquivo `main.js`

No trecho de código acima, Figura 3.2, vemos a criação de duas constantes, na linha 35, `Cc` (alias para Componets.classes) e `Ci` (alias para Components.interfaces) para podermos acessar os objetos desses componentes. Na linha 36 é criada uma variável, `globalMM`, apontando para o objeto do gerenciador de mensagens global do Firefox. E na linha 41 é criada uma função “`addMessageListener`” que acessa a função de mesmo nome dentro do gerenciador de mensagens do Firefox, linha 43, para registrar a mensagem passada (`name`) e a função (`func`) que deve ser executada ao ser recebida essa mensagem.

Neste ponto, a nova proposta de implementação sugerida por este trabalho nos levou a acrescentar ao gerenciador de mensagens uma escuta para a mensagem “`GotLoadEvent`” (no *Parent Process*), como observado na Figura 3.2, linha 57. Esta escuta foi adicionada para setar na notificação mostrada quando o mecanismo de Canvas Fingerprint for detectado. Assim, se a variável `isCF` (is Canvas Fingerprint) for verdadeira, executa a função `notity` da linha 61 da Figura

3.2 para fazer essa alteração da notificação. A função que envia esta mensagem e aciona a execução dessa escuta de mensagem (no gerenciador de mensagens) está definida no arquivo `frame.js`, mostrado mais abaixo.

Na mesma figura, mais abaixo na linha 69, tem o código da função "`addMessageListener`" que adiciona ao gerenciador de mensagens a espera pela mensagem "`canvasBlocker-check`" que, quando ela for recebida, executa a função `check` definida através do *include* da linha 11. Da mesma forma, na linha 74 é encontrado o código da função "`addMessageListener`" que adiciona ao gerenciador de mensagens a espera pela mensagem "`canvasBlocker-notify`" que, quando esta mensagem for recebida, executa a função `notify` que foi definida no *include* da linha 11. E assim por diante, outras escutas de mensagens foram adicionadas ao gerenciador de mensagens do Firefox originalmente para o funcionamento da extensão, mas que não serão mostradas aqui e podem ser vistas nos anexos.

E agora, para completar o processo de utilização do gerenciador de mensagens global do Firefox pela extensão, uma parte do arquivo "`frame.js`", principal arquivo carregado pelo *Content Process* é mostrado na Figura 3.3.

Adicional à implementação proposta neste estudo, na Figura 3.3, linha 76 do código, observamos a inclusão de uma escuta para a ocorrência do evento "`DOMContentLoaded`", no *Content Process*. Este evento ocorre quando o documento HTML for completamente carregado e analisado, sem aguardar por folhas de estilo, imagens, e subframes para encerrar o carregamento. Assim, quando este evento ocorre é executado, dentre outros, o comando da linha 80, ou seja, é enviada a mensagem "`GotLoadEvent`" (com alguns parâmetros) para o gerenciador de mensagens. Este, por sua vez, envia a mensagem para o *Parent Process* para executar os comando correspondentes a essa mensagem, ou seja, executa a alteração da notificação que mostra quando um Canvas Fingerprint for detectado (definição da escuta de mensagem mostrada mais acima no arquivo `main.js`).

Já, na linha 70, do código da Figura 3.3, existe a adição de uma escuta para a ocorrência do evento "`DOMWindowCreated`" que, quando ocorre, chama a função "`interceptWrapper`" que por sua vez chama a função "`intercept`" (linha 63, Figura 3.3). O evento "`DOMWindowCreated`" acontece quando o objeto *Window* está sendo criado, logo no início do carregamento de cada página web.

Assim, em resumo, a extensão faz alterações nos protótipos (*prototypes*) de algumas funções de Canvas neste momento, ou seja, antes delas serem utilizadas na página para serem utilizadas com suas respectivas alterações.

```
...
11    const {intercept} = require("../lib/intercept.js");
12    const {ask} = require("../lib/askForPermission.js");
...
17    function check(message){
18        if (enabled){
19            var status = sendSyncMessage(
20                "canvasBlocker-check",
21                message
22            );
23            return status[0];
24        }
25        else {
26            return {type: [], mode: "allow"};
27        }
28    }
29    function askWrapper(data){
30        return ask(data, {
31            _: function(token){
32                return sendSyncMessage(
33                    "canvasBlocker-translate",
34                    token
35                )[0];
36            },
37            prefs: function(name){
38                return sendSyncMessage(
39                    "canvasBlocker-pref-get",
40                    name
41                )[0];
42            }
43        });
44    }
45    function notify(data){
46        sendAsyncMessage("canvasBlocker-notify", data);
47    }
...
58    function interceptWrapper(ev){
59        if (enabled){
60
61            var window = ev.target.defaultView;
62
63            intercept(
64                {subject: window},
65                {check, ask: askWrapper, notify}
66            );
67        }
68    }
69
70    addEventListener("DOMWindowCreated", interceptWrapper);
...
75  //  EJr:
76    addEventListener("DOMContentLoaded", function (ev) {
77        var window = ev.target.defaultView;
78        var isCF = window.localStorage.getItem("jCanvasFingerprint");
79        var error = new Error();
80        sendAsyncMessage("GotLoadEvent", {url: window.location.href, errorStack: error.stack, isCF: isCF});
81    }, false);
...
```

Figura 3.3 - Trecho de código do arquivo `frame.js`

Na implementação original da extensão, é na função "`intercept`" que acontece essa alteração dos protótipos do mecanismo de Canvas. Essa função é carregada na linha 11 do código da Figura 3.3 no objeto `intercept`, que é uma função que será chamada na linha 63.

Observe na Figura 3.3 que são passados como parâmetros da função `intercept` o objeto `window`, a função `check` definida na linha 17, a função `ask` carregada com o *include* da linha 12 e a função `notify` definida na linha 45. O que é bom observar é que estamos no *Content Process* e que na função `notify`, na linha 46, é utilizado o mecanismo de gerenciamento de mensagens global do Firefox para enviar a mensagem "`canvasBlocker-notify`" (e a variável `data`) ao *Parent Process*. Observe também que o envio de mensagens também é utilizado para se comunicar com o *Parent Process* nas funções `check` (definida na linha 17) e na função `askWrapper` (definida na linha 29, Figura 3.3).

Desta forma, na função `intercept` (definida no arquivo `intercept.js`) são percorridos todos os objetos definidos na variável `changedFunctions` (definida no arquivo `modifiedAPI.js` e carregado na linha 8 do trecho de código da Figura 3.4, arquivo `intercept.js`). Neste mesmo arquivo, na linha 15 é armazenada na variável `original` a função original do protótipo. Mais abaixo, ainda durante a ocorrência do evento "`DOMWindowCreated`", os objetos dos protótipos das funções de Canvas são alterados na linha 17. Nessa alteração, o mais importante é observar a alteração da função `get`, que acontece na linha 23 do código da Figura 3.4, para cada um dos protótipos. Esta alteração tem algumas ações que acontecem dependendo das variáveis `status.type` e do `status.mode`, que podem retornar uma função falsa, retornar a função verdadeira, retornar uma função indefinida e/ou mostrar uma mensagem de alerta quando da detecção da utilização do mecanismo de Canvas.

Por exemplo, nos nossos experimentos, não testamos todas as possibilidades e combinações das preferências, mas deixamos sempre o parâmetro "`Block mode`" setado para "`fake readout API`", o "`ask only once`" marcado e o "`Show notifications`" marcado. Desta forma, quando entramos no site <http://ingresso.ifrs.edu.br>, por exemplo, recebemos um aviso visual (em amarelo) como mostrado na Figura 3.5.

Quando o aviso (amarelo) for mostrado na tela, como observado na Figura 3.5, podemos perceber que foi detectada a utilização do mecanismo de Canvas do HTML5. Neste aviso também observa-se alguns botões que ao clicar neles podemos ver a URL que chamou a função de Canvas, ver o comando que chamou a função, ignorar o domínio, adicionar a URL na lista branca ou na lista negra, ou ainda,

desabilitar esta notificação. Mas, como falado, estas funções não foram completamente testadas neste experimento, deixamos a configuração padrão citada anteriormente.

```
. . .
const {changedFunctions} = require("./modifiedAPI");
var apiNames = Object.keys(changedFunctions);
var undef;

exports.intercept = function intercept({subject: window}, {check, ask, notify}){
    apiNames.forEach(function(name){
        var changedFunction = changedFunctions[name];
        var original = window.wrappedJSObject[changedFunction.object].prototype[name];

        Object.defineProperty(
            window.wrappedJSObject[changedFunction.object].prototype,
            name,
            {
                enumerable: true,
                configureable: false,
                get: function(){
                    if (!window.location.href){
                        return undef;
                    }
                    EJr:
                    var isCF = window.localStorage.getItem("jCanvasFingerprint");

                    var error = new Error();
                    var status = check({url: window.location.href, errorStack: error.stack});
                    if (status.type.indexOf(changedFunction.type) !== -1){

                        if (status.mode === "ask"){
                            status.mode = ask({window: window, type: changedFunction.type, canvas: this, errorStack: error.stack});
                        }
                        switch (status.mode){
                            case "allow":
                                return original;
                            case "fake":
                                EJr:
                                notify({url: window.location.href, errorStack: error.stack, isCF: isCF}, window);
                                Original:
                                notify({url: window.location.href, errorStack: error.stack}, window);
                                return changedFunction.fake || undef;
                            //case "block":
                            default:
                                return undef;
                        }
                    }
                    EJr:
                    else if (changedFunction.type === "junior") {

                        return changedFunction.fake || undef;

                    }
                    else {
                        return original;
                    }
                }
            }
        );
    });
};
. . .
```

Figura 3.4 - Trecho de código do arquivo intercept.js

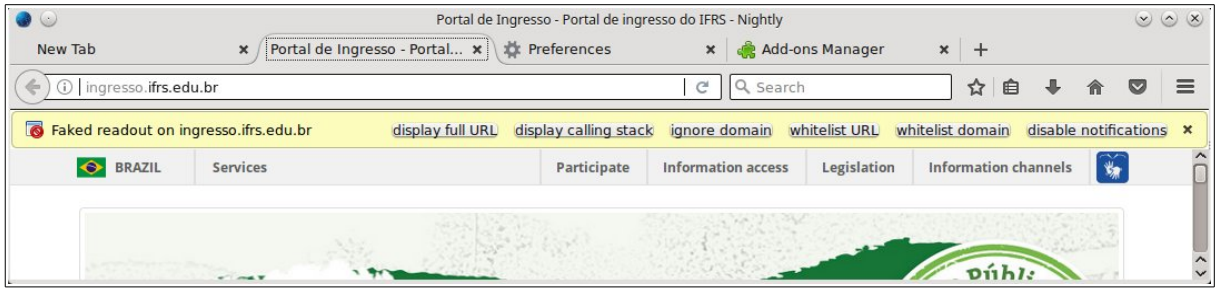

Figura 3.5 - Exemplo de alerta produzido pela extensão Canvas Blocker quando detectado Canvas (faixa amarela no topo da página com os botões de ações).

No trecho de código da Figura 3.4, a partir da linha 34, podemos observar que estando setado o modo "*ask*", é perguntado ao usuário o que ele quer fazer nos próximos acessos a este domínio ou URL. Em seguida, com a variável `status.mode` já setada (na linha 31 ou na linha 35), é escolhida a função a ser utilizada (linha 37). Assim, esta retorna a função original caso o modo seja "*allow*", retorna uma função falsa (função original, mas com alguns procedimentos que fazem modificações randômicas na imagem original do Canvas) caso o modo seja "*fake*" ou, nos outros casos, inclusive no modo "*block*", não retorna nada, bloqueando o mecanismo de Canvas.

Ainda, na Figura 3.4, podemos observar algumas alterações no código (propostas por esse trabalho) para avisar especificamente a detecção de Canvas Fingerprint. Para isso, observe que na linha 28 busca-se o valor da variável `jCanvasFingerprint` no `localStorage` do objeto `window` para saber se foi detectado Canvas Fingerprint. Este valor é armazenado na variável `isCF` e passado na função `notify` (Figura 3.4 linhas 42) para setar o tipo de aviso que irá aparecer, ou seja, avisando se detectou ou não Canvas Fingerprint. Este novo aviso aparecerá juntamente com os avisos de exemplo já mostrados na Figura 3.5.

Na linha 52 da Figura 3.4 pode-se observar outra modificação no código levada pela proposta deste trabalho. Para diferenciar as alterações dos protótipos que já estavam na extensão das alterações que foram introduzidas por esta proposta, definimos a variável `changedFunction.type` como `junior`. Desta forma, para a extensão continuar funcionando, foi adicionada essa nova condição que, apesar de parecer retornar a função "fake" (este é apenas o nome), retorna a função original do protótipo.

Antes de falarmos nas principais modificações do arquivo "`modifiedAPI.js`", precisamos citar a inclusão de um novo arquivo fonte, o "`zPlus.js`". Com a nova proposta de implementação, adicionou-se este arquivo aos fontes contendo dois conjuntos de funções auxiliares. Um visando auxiliar os logs dos dados que acontecem em diversas partes do programa, logs nos arquivos e no terminal, recolhendo dados que serão posteriormente analisados, dos quais os resultados serão tirados. O outro conjunto de funções foi baixado do site <http://pajhome.org.uk/site/legal.html>, são tilizados para automatizar a geração de um MD5 sobre uma string passada. Este MD5 é utilizado como nome de algumas variáveis dentro do arquivo "`modifiedAPI.js`".

Com isso, podemos ver a seguir mais detalhes das principais modificações da extensão e suas implicações no funcionamento do mecanismo de detecção.

3.2 Principais Mudanças no Mecanismo de Bloqueio

Algumas das modificações que se fizeram necessárias para a nova sugestão de implementação que este estudo propõe já foram mencionadas na seção anterior. Iremos discutir aqui os dois módulos que sofreram as modificações mais significativas devido à nova proposta de implementação. Estes são os arquivos "`modifiedAPI.js`" e "`notifications.js`".

Vejamos inicialmente o código do arquivo "`notifications.js`", mostrado abaixo, e as alterações que foram necessárias para mostrar o aviso de detecção de Canvas Fingerprint para esta solução estudada.

O código do arquivo "`notifications.js`", mostrado na Figura 3.6, exibe e atualiza a mensagem que pode ser vista no exemplo da Figura 3.5. Ainda, com a mensagem, são mostrados alguns botões que, ao serem clicados, acionam um aviso pop-up com algumas informações sobre a detecção do mecanismos de Canvas ou executam algum procedimento para alterar configurações da extensão. Assim, se o objeto `notification` existir (`if` linha 47) os dados serão atualizados a partir da linha 49, caso contrário ele será criado a partir da linha 62, na Figura 3.6.

Por isso, inicialmente setamos uma nova propriedade para o objeto `notification`, a propriedade `notification.isCF`, que é setada na linha 161 quando o objeto é criado ou alterada na linha 58 se o parâmetro `isCF` recebido pela função `notify` for verdadeiro (linha 12).

Para mostrar esse aviso adicionamos um botão com label "`Is Canvas Fingerprint?`", implementado da linha 63 à linha 80. Ao ser pressionado, este botão testa, na linha 70, a propriedade `notification.isCF`. Se tiver o valor verdadeiro seta a variável `isCF_msg` com a mensagem de detecção de Canvas Fingerprint (linha 71). Se tiver o valor falso seta a mensagem avisando que não é Canvas Fingerprint (linha 74). Em seguida, mostra a mensagem para o usuário na linha 76.

```
...
12  exports.notify = function({url, errorStack, isCF, isToShow}, {lists, notificationPref, _, browser, window}){
...
45      var notifyBox = tabBrowser.getNotificationBox(browser);
46      var notification = notifyBox.getNotificationWithValue("fake-readout");
47      if (notification){
48  //      EJr:
49          if (url != "about:blank") {
50              notification.label = message;
51              notification.url = url;
52              notification.domain = domain;
53              notification.callingStackMsg = callingStackMsg;
54          }
55
56  //      EJr:
57          if (isCF) {
58              notification.isCF = isCF;
59          }
60      }
61      else {
62          var buttons = [
63              {
64                  label: "Is Canvas Fingerprint?",
65                  accessKey: "",
66                  callback: function(){
67
68  //                  EJr:
69                      var isCF = notification.isCF;
70                      if (notification.isCF) {
71                          isCF_msg = "CAUTION: Canvas Fingerprint Detected!";
72                      }
73                      else {
74                          isCF_msg = "There is Canvas,\n but there isn't Canvas Fingerprint!";
75                      }
76                      window.alert(isCF_msg);
77
78                      throw new Error("Do not close notification.");
79                  }
80              },
...
148         var priority = notifyBox.PRIORITY_WARNING_MEDIUM;
149         notification = notifyBox.appendNotification(
150             message,
151             "fake-readout",
152             "chrome://browser/skin/Info.png",
153             priority,
154             buttons
155         );
156         notification.url = url;
157         notification.domain = domain;
158         notification.callingStackMsg = callingStackMsg;
159
160 //      EJr:
161         notification.isCF = isCF;
162
163     }
...
```

Figura 3.6 - Trecho de código do arquivo notifications.js

Cabe salientar ainda no código da Figura 3.6 dois pontos. O primeiro (linha 47) é quando o objeto `notification` já existe, neste caso a propriedade `notification.isCF` somente é alterada se o parâmetro `isCF` for verdadeiro (linha 57), ou seja, uma vez detectado Canvas Fingerprint continua o aviso de detecção, não altera. O outro ponto a ser mencionado ocorre também quando o objeto `notification` já existe, onde somente são alteradas as outras propriedades deste objeto quando o parâmetro `url` passado for diferente de "`about:blank`" (linha 49), pois se for igual os valores são errôneos ou nulos e não se deseja sobrescrever os antigos valores com valores incorretos.

Desta forma, vejamos na Figura 3.7 como este novo aviso (em amarelo) será mostrado para o usuário. Ele é parecido com o mostrado na Figura 3.5 acima, acrescido do primeiro botão que é mostrado com a inscrição "`Is Canvas Fingerprint?`". Quando pressionado mostra um aviso dizendo se detectou ou não

Canvas Fingerprint, como explicado no código da Figura 3.6, ou seja, avisa ao usuário se o site que está sendo visitado fez sua identificação (ou não) através do mecanismo de Canvas Fingerprinting.

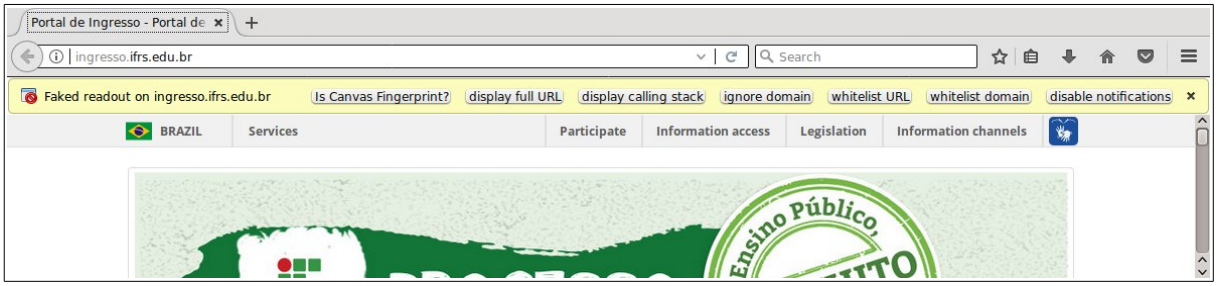

Figura 3.7 - Exemplo de alerta produzido pela extensão Canvas Blocker com as modificações avisando Canvas Fingerprint (faixa amarela no topo da página é o aviso, botão adicional “Is Canvas Fingerprint?”).

Após serem vistas as alterações nos procedimentos que mostram a notificação do aviso de detecção, seguiremos nossa análise com o arquivo “`modifiedAPI.js`”. Neste, as ideias iniciais basearam-se na pesquisa realizada por Acar et al. (2014) onde foram analisados os primeiros 100.000 links do site Alexa. Eles desenvolveram um rastreador baseado num navegador modificado e executaram rastreamentos exploratórios. Em seus apontamentos sugerem os seguintes procedimentos para eliminar falsos positivos e melhorar a detecção de Canvas Fingerprinting:

- Devem estar presentes ambas as chamadas para os métodos `ToDataURL` e `fillText` (ou `strokeText`), e ambas virem da mesma URL;

- A imagem de Canvas lida pelo script deve conter mais de uma cor e seu tamanho agregado deve ser maior que 16x16 *pixels*; e

- A imagem não deve ser requisitada em um formato compactado, como JPEG por exemplo.

Desta forma, as primeiras alterações no programa Canvas Blocker foram feitas para alcançar esse padrão de detecção. Em seguida, verificamos quais os resultados que obteríamos comparando com a implementação original do Canvas

Blocker.

Assim, originalmente, no arquivo “`modifiedAPI.js`” já estavam presentes as alterações dos protótipos dos objetos `getContext`, `toDataURL`, `toBlob`, `mozGetAsFile`, `getImageData` e `readPixels`. Para conseguir o que este estudo se propõe, fizemos mais algumas alterações nos protótipos dos objetos `toDataURL` e `getImageData`. Além disso, acrescentou-se alteração nos protótipos dos objetos `fillText`, `strokeText`, `fillRect` e `arc`. Vejamos os códigos fontes a seguir que mostram algumas dessas alterações.

```
...
71    exports.changedFunctions = {
...
251 //    EJr:
252       strokeText: {
253           type: "junior",
254           object: "CanvasRenderingContext2D",
255           fake: function strokeText(text, x, y, maxWidth){
256               var window = getWindow(this.canvas);
257               var str_msg=this.canvas.baseURI + '-width:' + this.canvas.width + '-height:' + this.canvas.height;
258               var pg_label = zPlus.calcMD5_2(str_msg);
259               if ( parseInt(this.canvas.width) > 16 &&  parseInt(this.canvas.height) > 16 ) {
260                   var pg_valor = parseInt(window.localStorage.getItem(pg_label));
261                   if ( pg_valor > 0 )
262                       pg_valor += 1;
263                   else
264                       pg_valor = 1;
265                   window.localStorage.setItem(pg_label, pg_valor);
266               }
267               return window.CanvasRenderingContext2D.prototype.strokeText.apply(this, arguments);
268           }
269       },
...
```

Figura 3.8 - Trecho de código do arquivo modifiedAPI.js - strokeText

Neste trecho de código (Figura 3.8), a partir da linha 252, observamos a definição do protótipo da função de Canvas `strokeText`. Toda vez que este procedimento for chamado e a imagem de Canvas for maior que 16x16 *pixels* (linha 259) é incrementada a variável global `pg_label`. O nome dessa variável é um MD5 gerado na linha 258 (função definida no include `zPlus.js`) a partir da string da linha 257. Esta, por sua vez, é formada pela URL da página e os tamanhos de largura e altura do Canvas. Desta forma, gera um nome específico para cada página web que está sendo acessada e de acordo com o tamanho do Canvas. Tendo o nome da variável, na linha 265 é setado o novo valor incrementado.

Observe ainda que, apesar desse procedimento que incrementa a variável global, na Figura 3.8, na linha 267 o resultado da execução da função original do procedimento `strokeText` é retornado para quem a chamou.

```
...
194 //      EJr:
195         fillText: {
196             type: "junior",
197             object: "CanvasRenderingContext2D",
198             fake: function fillText(text, x, y, maxWidth){
199                 var window = getWindow(this.canvas);
200                 var str_msg=this.canvas.baseURI + '-width:' + this.canvas.width + '-height:' + this.canvas.height;
201                 var pg_label = zPlus.calcMD5 2(str msg);
...
236                 if (   parseInt(this.canvas.width) > 16
237                     && parseInt(this.canvas.height) > 16
238                     && ((myArray != null && myArray.length != 1) || myString != "")
239                 ) {
240                     var pg_valor = parseInt(window.localStorage.getItem(pg_label));
241                     if ( pg_valor > 0 )
242                         pg_valor += 1;
243                     else
244                         pg_valor = 1;
245                     window.localStorage.setItem(pg_label, pg_valor);
246                 }
247                 return window.CanvasRenderingContext2D.prototype.fillText.apply(this, arguments);
248             }
249         },
...
```

Figura 3.9 - Trecho de código do arquivo modifiedAPI.js - fillText

Na Figura 3.9 observamos a alteração do protótipo da função de Canvas `fillText`. Da mesma forma que no código da função `strokeText` (Figura 3.8), na Figura 3.9, toda vez que o procedimento `fillText` for chamado e a imagem de Canvas for maior que 16x16 *pixels* (linha 236 e 237) é fixado um novo valor para a variável global `pg_label` (linha 245). Esta variável é um MD5 (gerada na linha 201) sobre a string fixada na linha 200, diferente para cada página, mas igual quando for chamada pela mesma página e com o mesmo tamanho de imagem de Canvas.

Ainda, da mesma forma que na função `strokeText`, na Figura 3.9, na linha 247 o resultado da execução da função original do procedimento `fillText` é retornado para quem a chamou.

Cabe salientar, que a variável fixada (ou incrementada) nas funções `fillText` e `strokeText` é utilizada para contar o número de vezes que estes procedimentos são chamados, pois é uma das condições para detectar Canvas Fingerprint definida em Acar et al. (2014) e citada anteriormente.

A título de informação, logo no início do arquivo "`modifiedAPI.js`" temos a função "`getFakeCanvas`". Ela já fazia parte da extensão e é nela que, de forma randômica, é gerada uma imagem de Canvas "`fake`", ou seja, com algumas alterações randômicas nos dados da imagem de Canvas, gerando assim uma nova imagem um tanto quanto modificada cada vez que for chamada.

Um exemplo da imagem original e da imagem fake pode ser observado na Figura 3.10 abaixo. Aparentemente são iguais, mas a imagem "fake" está diferente da original, ela apresenta cores um pouco mais escuras. Contudo, não vamos entrar em discussão sobre a alteração randômica que essa função faz na imagem, pois

não é o foco do trabalho.

	
Imagem Original	Imagem Fake

Figura 3.10 - Exemplo de imagem original e fake geradas pela extensão.

Assim, vejamos o código do protótipo da função `toDataURL` mostrado abaixo. Não nos esqueçamos que, de forma similar, os comandos da linha 84 e linha 121 da Figura 3.11 já existiam no código original da extensão, mas foram adaptados para serem utilizados na nova proposta. E, que na linha 84 da Figura 3.11 é chamada a função `getFakeCanvas` mencionada acima que gera a imagem "fake" mostrada no exemplo da Figura 3.10.

Ainda, no código da Figura 3.11, na linha 83, coletamos na variável `dataURL_orig` a execução da função `toDataURL` original. E, depois de conseguir o MD5 da variável que conta a quantidade de vezes que chamou `fillText` e `strokeText` nas linhas 85 e 86, armazenamos o valor desta na variável `pg_valor` (linha 87).

Entre as linhas 98 e 110, da Figura 3.11, calculamos a quantidade de cores que a imagem em questão tem. Se a mesma contiver mais de uma cor, setamos a variável `nColors` (linha 106) como verdadeira.

Depois de setar estas duas variáveis (`pg_valor` e `nColors`), verificamos se as condições para detectar Canvas Fingerprint definida em Acar et al. (2014) aparecem nesse Canvas. Para isso, estamos dentro da função `toDataURL` (já atendemos essa condição) verificamos se a quantidade de vezes que chamou `fillText` e `strokeText` é maior que um (linha 93), verificamos se a imagem de Canvas tem dimensões maiores que 16 *pixels* (linha 96) e verificamos se existe mais que uma cor (linha 111). Ocorrendo todas essas condições temos uma situação de detecção de Canvas Fingerprint, então removemos a variável `pg_label` do `localStorage` (linha 112) e setamos a variável "`jCanvasFingerprint`" em `localStorage` como verdadeira (linha 113).

Depois, vale lembrar, tendo a variável "`jCanvasFingerprint`" setada como verdadeira, a informação da ocorrência de Canvas Fingerprint é sinalizada na mensagem de aviso quando o evento "`DOMContentLoaded`" ocorrer, como já

mencionado anteriormente.

```
...
76      toDataURL: {
77          type: "readout",
78          object: "HTMLCanvasElement",
79          fake: function toDataURL(imageType){
80              var window = getWindow(this);
81
82  //          EJr:
83              var dataURL_orig = window.HTMLCanvasElement.prototype.toDataURL.apply(this);
84              var dataURL_fake = window.HTMLCanvasElement.prototype.toDataURL.apply(getFakeCanvas(window, this), arguments);
85              var str_msg=this.baseURI + '-width:' + this.width + '-height:' + this.height;
86              var pg_label = zPlus.calcMD5_2(str_msg);
87              var pg_valor = parseInt(window.localStorage.getItem(pg_label));
...
92              if (   dataURL_orig.length > 995
93                  && pg_valor > 0
94                  && (pg_valor_not > 0 || (pg_valor_not < 1 && pg_valor_is < 1))
95              ) {
96                  if ( parseInt(this.width) > 16 &&  parseInt(this.height) > 16 ) {
97                      if (imageType != "junior") {
98                          var ctx = window.HTMLCanvasElement.prototype.getContext.call(this, "2d");
99                          var imageData = ctx.getImageData(0,0,this.width, this.height,"junior");
100                         var data = imageData.data;
101                         var nA=-7, nB=-7;
102                         var nColors = false;
103                         for (var i = 0; (i < data.length && ((nA == -7) || (nB == -7) || (nA == nB))); i += 4) {
104                             var nA = (data[i] + data[i + 1] + data[i + 2] + data[i + 3]);
105                             if ((nA != -7) && (nB != -7) && (nA != nB)) {
106                                 nColors = true;
107                             } else {
108                                 if (nA > -7) nB = nA;
109                             }
110                         }
111                         if ( nColors ) {
112                             window.localStorage.removeItem(pg_label);
113                             window.localStorage.setItem('jCanvasFingerprint', true);
114                         }
115                     }
116                 }
117             }
118             if (imageType == "junior") {
119                 return dataURL_orig;
120             } else {
121                 return dataURL_fake;
122             }
123         }
124     },
...
```

Figura 3.11 - Trecho de código do arquivo modifiedAPI.js - toDataURL

Ainda sobre este código, algumas vezes precisamos executar a função original `toDataURL` e não a fake. Então, passamos na variável `imageType` o valor "`junior`" e fazemos as verificações, no código da Figura 3.11, na linha 97, para não identificar como Canvas Fingerprint essa chamada e na linha 118 para retornar a execução da função original.

Como já mencionado, fez-se necessário alterar outros protótipos do HTML5, além dos já alterados pela extensão, para conseguirmos fazer a detecção de Canvas Fingerprint. Estes códigos podem ser vistos em anexo, mas mencionaremos resumidamente as três principais modificações. Primeiro, na função `getImageData` (como na função `toDataURL`) precisamos chamá-la internamente e queríamos o valor original e não o fake, então acrescentamos um argumento que quando passado com o valor "`junior`" retorna a execução original. As duas outras modificações foram nos protótipos `fillRect` e `arc`. Nestes dois apenas

acrescentamos uma função que registra alguns dados nos logs quando estas são chamadas, para conseguirmos identificar o tipo de Canvas Fingerprint quando analisamos os dados coletados.

Então, depois das adaptações na extensão para detecção de Canvas Fingerprint, realizamos o acesso dos sites e a coleta de algumas informações sobre os 100.000 primeiros sites disponibilizados na relação Alexa.

Realizamos este experimento algumas vezes e a cada vezes analisamos os dados e refinamos a nossa proposta de alteração da extensão até a proposta final. Assim, os resultados nos levaram a fazer duas novas modificações para obter resultados mais precisos, com o intuito de reduzir os falsos negativos e reduzir os falsos positivos, como mostrado a seguir.

3.3 Reduzindo Falsos Negativos

Os primeiros resultados demonstraram vários falsos negativos. Observamos que a imagem de canvas é composta por quatro canais: vermelho, verde, azul e alfa. Constatamos que, para ver a existência de mais de uma cor, não se deve utilizar apenas os canais RGB (vermelho, verde e azul), mas devemos levar em conta o canal alfa também, pois ele é responsável por alterações de cor.

Por isso, a alteração no arquivo "`modifiedAPI.js`", no protótipo "`toDataURL`", deve levar em consideração os quatro canais de cores quando busca identificar se uma imagem possui duas cores ou mais, como observado na linha 104 da Figura 3.12. Assim, analisando os quatro canais de cores temos uma precisão maior na constatação das duas cores presentes nas imagens geradas e, desta forma, uma melhor detecção do Canvas Fingerprinting com redução dos falsos negativos.

```
...
76      toDataURL: {
77          type: "readout",
78          object: "HTMLCanvasElement",
79          fake: function toDataURL(imageType){
80              var window = getWindow(this);
81
82  //          EJr:
83              var dataURL_orig = window.HTMLCanvasElement.prototype.toDataURL.apply(this);
84              var dataURL_fake = window.HTMLCanvasElement.prototype.toDataURL.apply(getFakeCanvas(window, this), arguments);
85              var str_msg=this.baseURI + '-width:' + this.width + '-height:' + this.height;
86              var pg_label = zPlus.calcMD5_2(str_msg);
87              var pg_valor = parseInt(window.localStorage.getItem(pg_label));
88              var pg_label_is = zPlus.calcMD5_2(str_msg) + "isEmoji";
89              var pg_valor_is = parseInt(window.localStorage.getItem(pg_label_is));
90              var pg_label_not = zPlus.calcMD5_2(str_msg) + "notEmoji";
91              var pg_valor_not = parseInt(window.localStorage.getItem(pg_label_not));
92              if (   dataURL_orig.length > 995
93                  && pg_valor > 0
94                  && (pg_valor_not > 0 || (pg_valor_not < 1 && pg_valor_is < 1))
95              ) {
96                  if ( parseInt(this.width) > 16 &&  parseInt(this.height) > 16 ) {
97                      if (imageType != "junior") {
98                          var ctx = window.HTMLCanvasElement.prototype.getContext.call(this, "2d");
99                          var imageData = ctx.getImageData(0,0,this.width, this.height,"junior");
100                         var data = imageData.data;
101                         var nA=-7, nB=-7;
102                         var nColors = false;
103                         for (var i = 0; (i < data.length && ((nA == -7) || (nB == -7) || (nA == nB))); i += 4) {
104                             var nA = (data[i] + data[i + 1] + data[i + 2] + data[i + 3]);
105                             if ((nA != -7) && (nB != -7) && (nA != nB)) {
106                                 nColors = true;
107                             } else {
108                                 if (nA > -7) nB = nA;
109                             }
110                         }
111                         if ( nColors ) {
112                             window.localStorage.removeItem(pg_label);
113                             window.localStorage.setItem('jCanvasFingerprint', true);
114                         }
115                     }
116                 }
117             }
118             if (imageType == "junior") {
119                 return dataURL_orig;
120             } else {
121                 return dataURL_fake;
122             }
123         }
124     },
...
```

Figura 3.12 - Trecho de código do arquivo modifiedAPI.js - toDataURL completa

3.4 Reduzindo Falsos Positivos

Analisando as imagens geradas nas várias coletas, observamos também vários falsos positivos. Estes falsos positivos ocorrem pois, muitos dos sites populares atualmente, utilizam o WordPress e verificam o suporte de Emojis pelo navegador. Estas pequenas imagens (Emojis) são geradas utilizando o mecanismo de Canvas do HTML5 e por isso eram detectadas como Canvas Fingerprinting.

Para exemplificar, algumas das imagens geradas pelo WordPress são mostradas na Figura 3.13. A imagem (a), dois quadrados vazios, é uma imagem gerada num teste quando o navegador não suporta Emoji. As imagens (b), (c) e (d) são imagens geradas em um teste quando o navegador suporta Emoji.

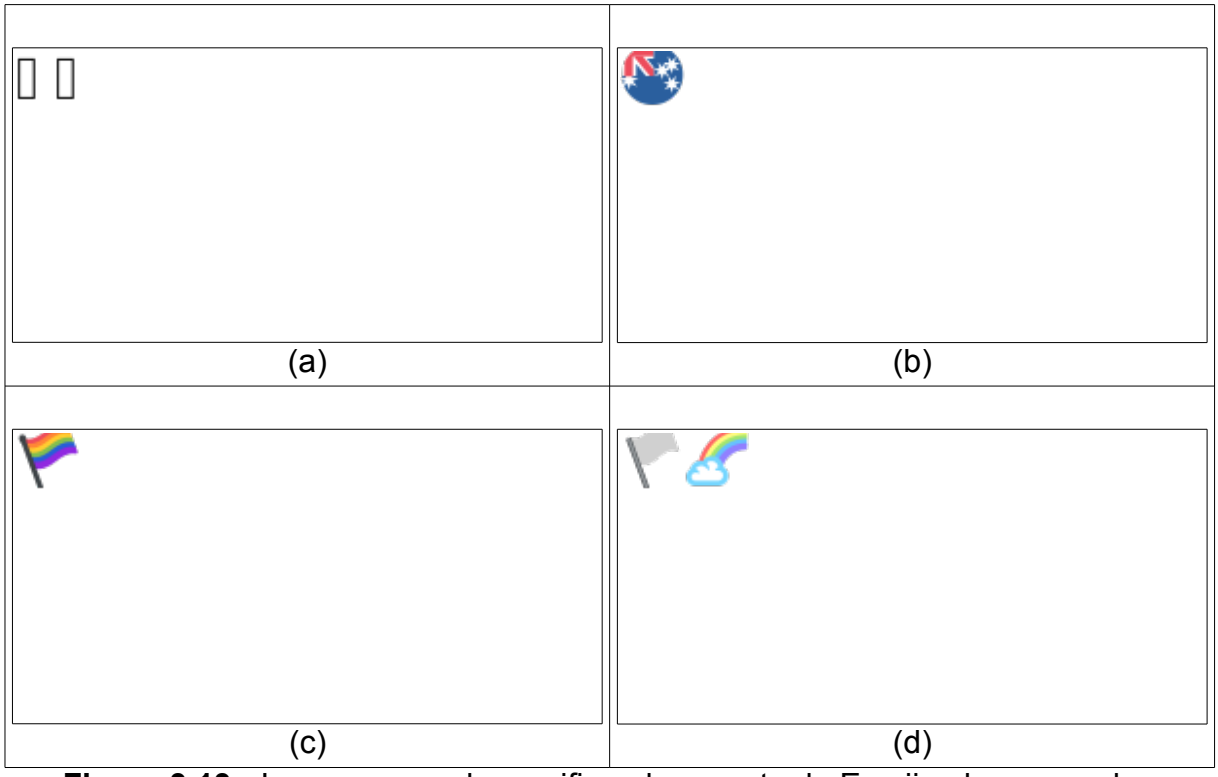

Figura 3.13 - Imagens geradas verificando suporte de Emoji pelo navegador.

Para imagens de erro (como é o caso da imagem (a) da Figura 3.13) um teste para verificar se seu tamanho, retornado pela função `toDataURL`, é menor que 995 descarta as falsas detecções de Canvas Fingerprint (teste encontrado no próprio WordPress, "`wp-emoji-loader.js`" e "`wp-emoji-loader.min.js`"). Assim, este teste foi implementado na função `toDataURL` e pode ser observado na Figura 3.12, linha 92.

Já, quando um Emoji é criado utilizando o mecanismos de Canvas, ele é escrito no Canvas utilizando a função `fillText`. Para isso, implementamos uma detecção de Emoji na função `fillText` que pode ser observada na Figura 3.14, a partir da linha 202.

Para isso, objetivando fazer uma detecção dos Emojis gerados pelo mecanismo de Canvas do HTML5, utilizamos uma expressão regular (regex ou *regular expression*) definida na Figura 3.14, linha 203. Esta expressão regular foi extraída do programa `emoji-regex`[6], que oferece uma expressão regular para

6 Programa `emoji-regex` encontrado em https://github.com/mathiasbynens/emoji-regex/. Este módulo pode ser instalado com o comando "`npm install emoji-regex`". Uma versão do mesmo foi instalada na pasta node_modules e foi entregue junto com os fontes da extensão Canvas Blocker modificada e disponibilizada segundo Anexo I.

identificar os símbolos de Emoji passados, conforme o padrão unicode dos mesmos.

Contudo, apesar da expressão regular ser bastante completa, ela não abrange todas as possibilidades de Emojis encontrados. Por isso, verificamos o texto passado para a função `fillText` em duas etapas. Ou ele é identificado como um único Emoji através dos testes das linhas 211 a 218 da Figura 3.14. Ou é identificado utilizando a expressão regular mencionada nos testes da linha 221 da Figura 3.14 (variável `myArray`, obtida na linha 204, deve ter apenas um elemento e variável `myString`, obtida na linha 205, deve ser vazia). Assim, passando em um destes testes contabilizamos, na variável `pg_label_is`, a detecção de um Emoji (linha 228). Caso contrário contabilizamos, na variável `pg_label_not`, que a string passada foi um texto e não um Emoji (linha 234).

```
...
194  //      EJr:
195          fillText: {
196              type: "junior",
197              object: "CanvasRenderingContext2D",
198              fake: function fillText(text, x, y, maxWidth){
199                  var window = getWindow(this.canvas);
200                  var str_msg=this.canvas.baseURI + '-width:' + this.canvas.width + '-height:' + this.canvas.height;
201                  var pg_label = zPlus.calcMD5_2(str_msg);
202                  var isText = text;
203                  var myRe1 = /... REGEX .../g;
204                  var myArray = myRe1.exec(isText);
205                  var myString = isText.replace(myRe1, '');
206                  var pg_label_is = zPlus.calcMD5_2(str_msg) + "isEmoji";
207                  var pg_valor_is = parseInt(window.localStorage.getItem(pg_label_is));
208                  var pg_label_not = zPlus.calcMD5_2(str_msg) + "notEmoji";
209                  var pg_valor_not = parseInt(window.localStorage.getItem(pg_label_not));
210                  if (
211                      ((     isText == String.fromCharCode( 55356, 57135 ) // D83C DF2F
212                          || isText == String.fromCharCode( 55356, 57221 ) // D83C DF85
213                          || isText == String.fromCharCode( 55356, 56806, 55356, 56826 ) // D83C DDE6 D83C DDFA
214                          || isText == String.fromCharCode( 55356, 57331, 55356, 57096 ) // D83C DFF3 D83C DF08
215                          || isText == String.fromCharCode( 55356, 57221, 55356, 57343 ) // D83C DF85 D83C DFFF
216                          || isText == String.fromCharCode( 55356, 57331, 65039, 8205, 55356, 57096 ) // D83C DFF3
217                          || isText == String.fromCharCode( 55357, 56835 ) // D83D DE03
218                          || isText == String.fromCharCode( 55358, 56631 ) // D83E DD37
219                       )
220                      )
221                      || (myArray != null && myArray.length == 1 && myString == "")
222                  ) {
223                      myString = "";
224                      if ( pg_valor_is > 0 )
225                          pg_valor_is += 1;
226                      else
227                          pg_valor_is = 1;
228                      window.localStorage.setItem(pg_label_is, pg_valor_is);
229                  } else {
230                      if ( pg_valor_not > 0 )
231                          pg_valor_not += 1;
232                      else
233                          pg_valor_not = 1;
234                      window.localStorage.setItem(pg_label_not, pg_valor_not);
235                  }
236                  if (   parseInt(this.canvas.width) > 16
237                      && parseInt(this.canvas.height) > 16
238                      && ((myArray != null && myArray.length != 1) || myString != "")
239                  ) {
240                      var pg_valor = parseInt(window.localStorage.getItem(pg_label));
241                      if ( pg_valor > 0 )
242                          pg_valor += 1;
243                      else
244                          pg_valor = 1;
245                      window.localStorage.setItem(pg_label, pg_valor);
246                  }
247                  return window.CanvasRenderingContext2D.prototype.fillText.apply(this, arguments);
248              }
249          },
...
```

Figura 3.14 - Trecho de código do arquivo modifiedAPI.js - fillText completa

Adicionalmente a estas conferências, passamos a fazer um teste na linha 238, na Figura 3.14, onde contabilizamos chamadas para a função `fillText` (na variável `pg_label`, linha 245) apenas quando não for um Emoji, caso contrário, não contabilizamos uma chamada a esta função.

Por fim, para não detectar a criação de um Emoji como um Canvas Fingerprint, na função `toDataURL`, pegamos os valores das variáveis `pg_valor_is` e `pg_valor_not` (Figura 3.12, nas linhas 87 a 91) e testamos, na linha 94, para deixar prosseguir na identificação de Canvas Fingerprint somente se não for um Emoji.

Desta forma, com todas essas verificações citadas, conseguimos reduzir os falsos positivos na detecção da utilização do mecanismo de Canvas Fingerprinting significativamente, levando em consideração a grande quantidade de sites, na atualidade, que utilizam o mecanismo de Canvas para desenhar Emojis.

4 RESULTADOS OBTIDOS

Aqui serão explanadas as definições feitas durante a implementação e explicações de como foram realizadas as visitas às páginas web para a coleta dos dados. Também serão mostrados a organização dos dados obtidos e sua análise, chegando nos resultados que serão apresentados.

4.1 Definições da Implementação

O sistema operacional Linux foi utilizado como base para todas as etapas, por isso, a maioria dos comandos e dicas aqui apresentadas serão voltados para este ambiente.

Durante a fase de implementação foi utilizada a versão Firefox Nightly[7] pois se adaptava melhor e fornecia um conjunto maior de ferramentas para a fase de desenvolvimento e testes que pretendíamos. Já na fase de coleta de dados, onde aplicamos a extensão sobre os 100.000 sites do ranque Alexa, utilizamos a versão padrão do navegador Firefox.

Como facilitador, existe um conjunto de páginas na Fundação Mozilla, que desenvolve as várias versões do Firefox, voltada especificamente para desenvolvedores e cujo endereço é *<https://developer.mozilla.org>*. Nesta área, existe um conjunto de ferramentas e tutoriais que nos ajudam a dar os primeiros passos para começar a desenvolver extensões para o Firefox e estão disponíveis no endereço *<https://developer.mozilla.org/en-US/Add-ons/SDK/Tutorials>*. Foi dessas páginas que tiramos a maioria dos comandos apresentados para a preparação do ambiente.

7 O Firefox Nightly pode ser baixado do endereço https://nightly.mozilla.org/

Durante o desenvolvimento precisamos instalar alguns pacotes e bibliotecas que nos auxiliaram no desenvolvimento. Pelo gerenciador de pacotes do Linux ou diretamente pelo site *<https://nodejs.org/> e <https://eclipse.org/>*, foi necessário instalar os pacotes: `nodejs`, `nodejs-legacy`, `npm` e `eclipse-wtp-webtools`. Estes são ferramentas que nos ajudarão na programação da extensão.

Depois de instalados estes pacotes, podemos ver a versão do node utilizando o comando "`node -v`". E, se necessário, para atualizar para a última versão utilize o comando "`sudo npm install -g n`" e, em seguida, execute o comando "`sudo n stable`" para setar a versão estável do mesmo.

Será necessária ainda a instalação do programa JPM com o comando "`sudo npm install jpm --global`". O programa JPM é baseado no Node-JS e permite testar, executar e empacotar extensões para o Firefox.

Depois das ferramentas de desenvolvimento instaladas, pode-se baixar os fontes da extensão Canvas Blocker no endereço *<https://github.com/kkapsner/CanvasBlocker>*. Neste site podemos encontrar várias versões da extensão. Inicialmente, começamos os estudos e experimentos com a versão 0.2.1. Entendemos o funcionamento da mesma e percebemos que utiliza a forma tradicional de processamento do Firefox, ou seja, trabalha com apenas um processo.

Contudo, para o desenvolvimento da pesquisa foi utilizada uma versão mais recente da extensão, a versão 0.3.0. Esta versão utiliza as últimas funcionalidades disponibilizadas pelo navegador Firefox, ou seja, utiliza a funcionalidade conhecida como E10s ou *Electrolysis* (Eletrólise). Em poucas palavras, a funcionalidade *Electrolysis*[8] (Eletrólise) no Firefox permite a utilização de múltiplos processos. Desta forma, as renderizações ou as execuções do conteúdo de uma página web são feitas pelos Processos Filho (C*hild Processes* ou *Content Process*). Estes, por sua vez, se comunicam com o Firefox pai (P*arent* Firefox), que seria o Processo Pai (P*arent Process* ou *Chrome Process*), através do protocolo IPDL (*IPC Protocol Definition Language*, onde IPC é *Inter-Process Communication*). Trabalhando assim, as duas principais vantagens são segurança (não permite acessar o conteúdo web diretamente) e performance (utiliza mais de um processo).

Desta forma, o Firefox permite a utilização de múltiplos processos e a implementação da extensão para detecção de Canvas Fingerprint desenvolvida

8 Maiores informações sobre essa funcionalidade podem ser encontradas no endereço *<https://wiki.mozilla.org/Electrolysis>*

neste trabalho foi feita em cima da da versão 0.3.0 do Canvas Blocker, para utilizar essa funcionalidade de múltiplos processos do Firefox.

Cabe ainda mencionar que, durante a implementação, na pasta onde tem os fontes da extensão, para executar e testar, deve-se utilizar o comando "`jpm run -b /usr/bin/firefox --debug`", para iniciar o Firefox normal. Se quiser iniciar o Firefox Nightly utilize o comando "`jpm run -b /caminho_ateh_pasta_do_firefox_nightly/firefox --debug`". A opção "`--debug`" abre a tela de debug do "Chrome Process" (ou "Parent Process"), ou seja, do processo pai. E, para abrir a janela de debug do "Content Process", na janela principal do navegador clicar nas opções > Developer > Browser Content Toolbox. Lembrando que esta opção apenas está disponível no Firefox Nightly.

Depois que a extensão modificada estiver pronta será necessário gerar um arquivo "*`.xpi`*", que é o pacote da extensão, utilizada para ser instalada no navegador Firefox. Assim, para gerar esse pacote, na pasta onde tem os fontes da extensão, deve-se executar o comando "`jpm xpi`".

Com o pacote da extensão pronto e instalado no Firefox, podemos abrir os sites propostos e coletar os dados. Para os nossos testes, foi tomada uma porção reduzida da relação de sites disponibilizada pelo Site Alexa (*http://www.alexa.com/topsites*). Desta listagem, disponibilizada diariamente no endereço eletrônico <*http://s3.amazonaws.com/alexa-static/top-1m.csv.zip*> contendo os "Top 1,000,000 websites" segundo informação do Site SEO Book, foram tomados apenas os primeiros 100.000 sites. Pois essa listagem já havia sido utilizada no trabalho desenvolvido por Acar et al. (2014) e pela empresa que gera a lista estar no mercado desde 1996, vinculada à empresa *amazon.com*, e ser uma das pioneiras no serviço de análise web e tráfego global da Internet, segundo o próprio site da empresa.

Por fim, durante a fase de coleta e análise dos dados, foi necessária a definição dos vários tipos de linhas de log de dados a serem coletadas durante a visita às páginas web. Por isso, para facilitar a análise e mineração dos dados coletados, as linhas de logs gravadas no arquivo de log foram identificadas cada uma com um código, este código define o tipo de dado que aquela linha continha. Assim, identificando os dados que necessitariam ser coletados durante as visitas aos sites, foram criados identificadores para as seguintes linhas de logs coletadas:

- **SRC_0** (ou **SRC0**): grava logs a partir do arquivo “`modifiedAPI.js`”. Loga informações das imagens de Canvas geradas e identificadas pela extensão. Dentre as informações coletadas, na função “`toDataURL`” loga a imagem de Canvas original e a imagem fake gerada, para futuras análises;

- **SRC_1** (ou **SRC1**): este tipo de linha de log é gerado dentro do arquivo “`modifiedAPI.js`”. Coleta dados quando são executados os procedimentos originais da extensão que identifica Canvas. Ou seja, quando é identificado Canvas da forma original da extensão esta linha de log é gravada com seus respectivos dados;

- **SRC_2** (ou **SRC2**): log gerado dentro do arquivo “`modifiedAPI.js`”, na função “`toDataURL`” quando identifica Canvas Fingerprint através do procedimento apresentado nesse trabalho, a nova proposta para a extensão. Ou seja, quando é identificada a utilização do mecanismo de Canvas Fingerprinting, esta linha é gravada no arquivo de log;

- **SRC_3** (ou **SRC3**): este tipo é gerado no arquivo “`check.js`”. Coleta a URL identificada pela extensão e a mensagem que será mostrada quando é identificado Canvas pela extensão. Utilizado na análise dos dados para identificação do tipo de Fingerprint; e

- **SRC_4** (ou **SRC4**): log que coleta dados dentro do arquivo “`modifiedAPI.js`”, nas funções “`fillText`”, “`fillRect`” e “`arc`”. Utilizado durante a análise dos dados para identificação do tipo de Fingerprint.

Os códigos fontes dos diferentes arquivos aqui mencionados, juntamente com os comandos que logam esses dados, não serão mostrados aqui. Contudo, para conhecimento, esses dados eram gravados em um arquivo texto utilizando a chamada de uma função com o seguinte formato “`zPlus.log_save(...dados...)`” e definida no arquivo fonte `zPlus.js`. Todavia, estes arquivos podem ser encontrados segundo as instruções encontradas

nos anexos, bem como os outros arquivos mencionados e dentro dos mesmos é possível visualizar a chamada da função que grava os dados nos logs.

Assim, depois de conhecer as principais definições para o entendimento da forma com que foi conduzida a pesquisa, podemos explanar como as visitas aos sites aconteceram, para coletar os dados de detecção da utilização do mecanismo de Canvas Fingerprinting.

4.2 Visitando as Páginas Web

Para agilizar o processo de visitação dos 100 mil sites do ranque da Alexa foram criadas 3 máquinas virtuais. Cada máquina virtual continha dois processadores com 6 núcleos cada, 16GB de RAM, duas conexões de rede utilizadas de forma simultânea e sistema operacional Xubuntu[9].

Em cada uma das máquinas o Firefox foi configurado com 3 perfis (canvas_01, canvas_02 e canvas_03). Em cada perfil, entrando na configuração avançada (para isso digitar no campo de navegação "`about:config`") e alterar as configurações segundo especificações encontradas na Figura 4.1.

```
xpinstall.signatures.required = false
browser.popups.showPopupBlocker = false
dom.popup_maximum = 20 (já estava com esse valor)
privacy.popups.showBrowserMessage = false
extensions.update.autoUpdateDefault = false
extensions.update.enabled = false
network.http.pipelining = true
network.http.pipelining.maxrequests = 32 (já estava com esse valor)
network.http.pipelining.ssl = true
network.http.proxy.pipelining = true
network.http.max-connections = 256 (já estava com esse valor)
network.http.max-connections-per-server = 16
```

Figura 4.1 - Configurações avançadas feitas no Firefox acessando, no campo de navegação, o endereço "about:config"

Para a abertura das páginas foi criado o script "`zRun_01_url_open.sh`". A

9 Sistema Operacional Linux, distribuição Xubuntu, encontrado em http://xubuntu.org/

sintaxe era a seguinte "`zRun_01_url_open.sh {file_name} {line_start} {line_end}`", onde `file_name` é o nome do arquivo que continha a relação de sites a ser visitada, `line_start` era a linha inicial e `line_end` a linha final da relação de sites. Com isso, por exemplo, a chamada ficaria assim "`zRun_01_url_open.sh 100mil_alexa.txt 1 10000`" para abrir os sites do arquivo passado que estivessem no intervalo da linha 1 até a linha 10 mil. Lembrando que, no arquivo passado, tinha um site por linha.

Assim, este script era executado em cada uma das máquinas virtuais. Para não sobrecarregar as mesmas eram passados intervalos de 10 mil sites por vez. Abriam-se no Firefox um total de 120 páginas por vez, 40 páginas em cada perfil. Esperava-se 7 minutos até as páginas carregarem completamente. Depois fechavam-se as três instâncias do Firefox (uma com cada perfil), esperava-se um minuto e, após, recomeçava-se o processo, até abrir todo o intervalo de sites passado.

O procedimento foi repetido nas três máquinas virtuais, com intervalos diferentes até serem abertos e processados todos os 100 mil sites do ranque da Alexa. Posteriormente, os dados foram organizados e minerados para serem analisados.

4.3 Organizando os Dados Obtidos

Os dados coletados foram minerados e analisados de duas formas diferentes. Uma através de relatório gerados pelos próprios scripts de análise. De outra forma, os dados coletados são inseridos no banco de dados PostgreSQL e confrontados com os dados obtidos através dos scripts criados de mineração automática.

Para a segunda forma de análise, foi utilizado o banco de dados PostgreSQL e algumas estruturas de dados/tabelas foram criadas. Para o script SQL que gera a tabela utilizada para armazenar os 100 mil sites do ranque Alexa foi utilizado o código da Figura 4.2.

```
-- Table: alexa
-- DROP TABLE alexa;
CREATE TABLE alexa
(
  id serial NOT NULL,
  domain character varying(100) NOT NULL,
  CONSTRAINT pk_alexa_id PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE alexa
  OWNER TO postgres;
```

Figura 4.2 - Script SQL que gera a tabela “alexa”

O script SQL que cria a tabela para os logs gerados pelas URL acessadas (linhas de log SRC1, SRC2, SRC3 e SRC4) é encontrado na Figura 4.3.

```
-- Table: url_log
-- DROP TABLE url_log;
CREATE TABLE url_log
(
  id serial NOT NULL,
  domain character varying(100) NOT NULL,
  domain_from character varying(30) NOT NULL,
  domain_link character varying(100) NOT NULL,
  "timestamp" character varying(30),
  date character varying(30),
  line_type character varying(10),
  detection character varying(10),
  function character varying(30),
  function_type text,
  referrer text,
  baseuri text,
  location_url text,
  CONSTRAINT pk_url_log_id PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE url_log
  OWNER TO postgres;
```

Figura 4.3 - Script SQL que gera a tabela “url_log”

E o script SQL utilizado para criar a tabela para os log's gerados das imagens de Canvas (linha de log SRC0) é encontrado na Figura 4.4.

```
-- Table: url_img
-- DROP TABLE url_img;
CREATE TABLE url_img
(
  id serial NOT NULL,
  domain character varying(100) NOT NULL,
  domain_from character varying(30) NOT NULL,
  domain_link character varying(100) NOT NULL,
  "timestamp" character varying(30),
  date character varying(30),
  line_type character varying(10),
  detection character varying(10),
  function character varying(30),
  function_type character varying(30),
  referrer text,
  baseuri text,
  width character varying(10),
  height character varying(10),
  img_size integer,
  img text,
  CONSTRAINT pk_url_img_id PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE url_img
  OWNER TO postgres;
```

Figura 4.4 - Script SQL que gera a tabela “url_img”

Depois das tabelas criadas e os dados inseridos, foram feitas pesquisas SQL sobre esses dados e os resultados confrontados com os relatórios obtidos pelos scripts de mineração automática que serão mencionados a seguir.

Para uma primeira análise, com scripts de mineração automática dos dados, foram criados scripts em Bash Script para analisar os dados obtidos e gerar relatórios. Para este fim foram criados dois scripts “`zRun_02_data_manipulation.sh`” e “`zRun_03_relatorios.sh`”.

Para o script “`zRun_02_data_manipulation.sh`” a sintaxe é da seguinte forma: “`zRun_02_data_manipulation.sh {coleta} {option} {line_start} {line_end}`”. Onde `coleta` é o nome da pasta onde estão os dados coletados (arquivos de log gerados), `option` seleciona o procedimento que se quer executar dentro do script e, `line_start` e `line_end`, definem o intervalo de sites que se quer processar. Assim, dependendo das opções passadas, pode-se:

- Gerar um arquivo com comandos SQL de inserção na tabela “`alexa`”, para inserir os domínios do intervalo de sites passados (com option = 1) ;

- Percorre todo o arquivo de log e gera comandos SQL de inserção nas tabelas “`url_log`” e “`url_img`”. Adicionalmente, extrai imagens de Canvas logadas, que estão no arquivo de log, gerando um arquivo JPG para cada imagem (com as opções 21, 22 e 23); e

- Com outra opção passada (opção 3) trata/identifica tipos específicos de Fingerprinting, conforme alguns padrões encontrados nos logs. Gerando uma lista com o site acessado e o tipo de Fingerprinting encontrado.

No script “`zRun_03_relatorios.sh`” a sintaxe é da seguinte forma: “`zRun_03_relatorios.sh {line_start} {line_end}`”. Onde `line_start` e `line_end` definem o intervalo de sites que se quer processar. Neste script são feitas várias comparações, os dados são processados e analisados gerando relatório de resultados que serão apresentados a seguir.

4.4 Apresentação dos Dados Obtidos

Primeiramente, vamos analisar os resultados obtidos através da detecção original da extensão Canvas Blocker. Estes resultados são apresentados a seguir.

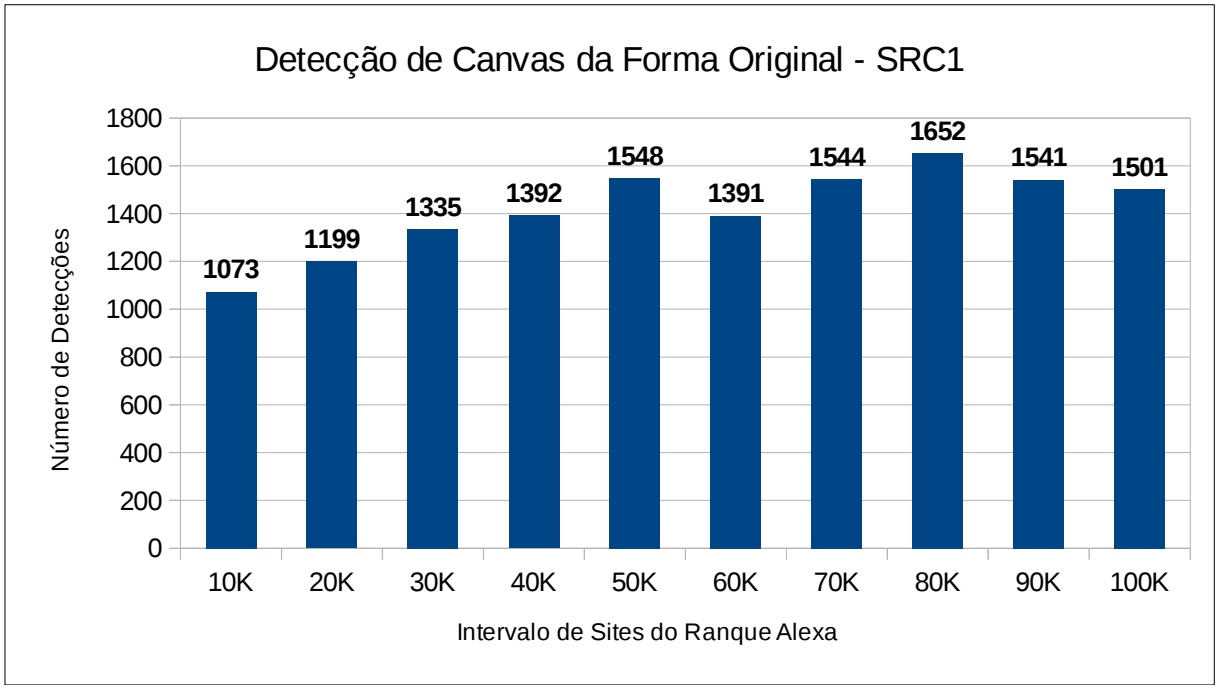

Figura 4.5 - Detecções de Canvas original da extensão Canvas Blocker (SRC1).

Na Figura 4.5 os valores são apresentados por colunas e correspondem ao intervalo de 10 mil em 10 mil sites. Por exemplo, na primeira coluna o intervalo é do site 1 ao site 10.000, na segunda coluna os valores correspondem do site 10.001 ao site 20.000, assim por diante. Esses intervalos correspondem à ordem de sites encontrada no ranque Alexa.

Totalizando o gráfico apresentado na Figura 4.5, dentre os 100 mil sites acessados do ranque da Alexa, temos que 14.176 sites utilizaram o mecanismo de Canvas e por isso foram detectados pela extensão Canvas Blocker. Falando em termos percentuais, estamos dizendo que 14,18% dos 100 mil sites já utilizam o novo mecanismo de Canvas disponibilizado pelo HTML5 em suas páginas web.

Apesar do esforço dispendido para identificar todas as linhas de log geradas e o seu respectivo site vinculado ao ranque Alexa, tivemos 3,44% (487) de sites, dos 14.174 identificados que utilizam Canvas, onde a linha de log não tem vinculação com um site do ranque. Pela investigação feita em algumas amostras, constatamos que os motivos principais são o redirecionamento para outra URL ou a abertura de outro site não vinculado ao ranque, como anúncios específicos que existem em alguns sites.

Da mesma forma que na figura anterior, na Figura 4.6 os valores são apresentados por colunas e correspondem a intervalos de 10 mil em 10 mil sites. Pode-se observar nessa figura as detecções SRC2 em cada faixa de valores, ou

seja, detecções de Canvas Fingerprint pela proposta deste trabalho.

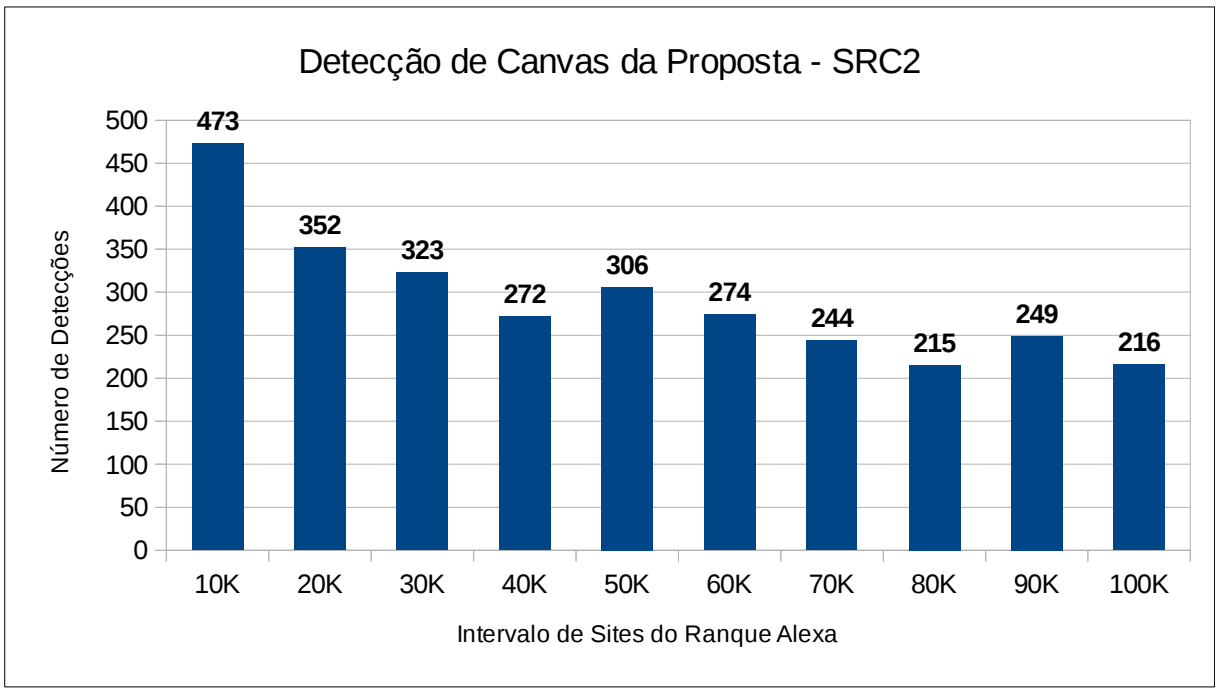

Figura 4.6 - Detecções de Canvas Fingerprinting, nova propostas da extensão Canvas Blocker (SRC2)

Assim, levando em consideração a nova proposta da extensão, podemos observar que a totalização dos valores apresentados na Figura 4.6 foi de 2.924, ou seja, 2,92% de sites, dos 100 mil sites do ranque da Alexa, utilizam essa forma de identificação de dispositivo/usuário.

Comparando com a Figura 4.5, na Figura 4.6 tivemos uma redução de 11.252 (11,25% sobre os 100 mil) sites identificados. Isso pois o SRC1 (forma original de identificação) identifica apenas Canvas, enquanto que o SRC2 (nova proposta) identifica características específicas de Canvas Fingerprinting.

Analisando a proposta SRC2, constatamos também a existência de linhas de log com domínios fora do ranque de sites da Alexa. Esse número corresponde a 3,59% (105 sites) dos 2.924 sites identificados com Canvas Fingerprinting. Da mesma forma que no anterior, investigando algumas amostras, constatamos que os motivos principais são o redirecionamento para outra URL (ou seja, a URL do ranque é antiga e o site adotou uma nova URL) ou houve a abertura de outro site, não vinculados ao ranque, como propagandas que são abertas dentro de alguns sites.

Outra análise que foi feita foi em relação aos tipos de Fingerprinting detectados. Na Figura 4.7 são mostrados esses tipos de Canvas Fingerprinting

identificados nessa pesquisa. Não são tipos novos, mas devido à quantidade encontrada se tornam relevantes.

Para fazer essa identificação, linhas de log foram geradas durante a visita aos sites do ranque Alexa. Essas linhas de log são identificadas com SRC4 e logaram-se características específicas do mecanismo de Fingerprinting, logando chamadas para as funções "`fillText`", "`fillRect`" e "`arc`".

Analisando os dados coletados, as imagens geradas e informações sobre o funcionamento das bibliotecas `FingerprintJS` e `FingerprintJS2`, separamos os tipos de Canvas Fingerprinting identificados em 4 tipos, como mostrados na Figura 4.7, que seriam:

- **fp1_js** - este tipo de fingerprint é gerado utilizando a biblioteca FingerprintJS sem modificações;
- **fp2_js** - este tipo de fingerprint é gerado utilizando a biblioteca FingerprintJS2 sem modificações;
- **fp1_like** - este tipo identificado utiliza a biblioteca FingerprintJS com modificações. Essas modificações podem ser no texto escrito na figura ou mesmo nas características da própria figura de canvas gerada; e
- **fp2_like** - este tipo identificado utiliza a biblioteca FingerprintJS2 com modificações. Essas modificações podem ser no texto escrito na figura ou mesmo nas características da própria figura de canvas gerada.

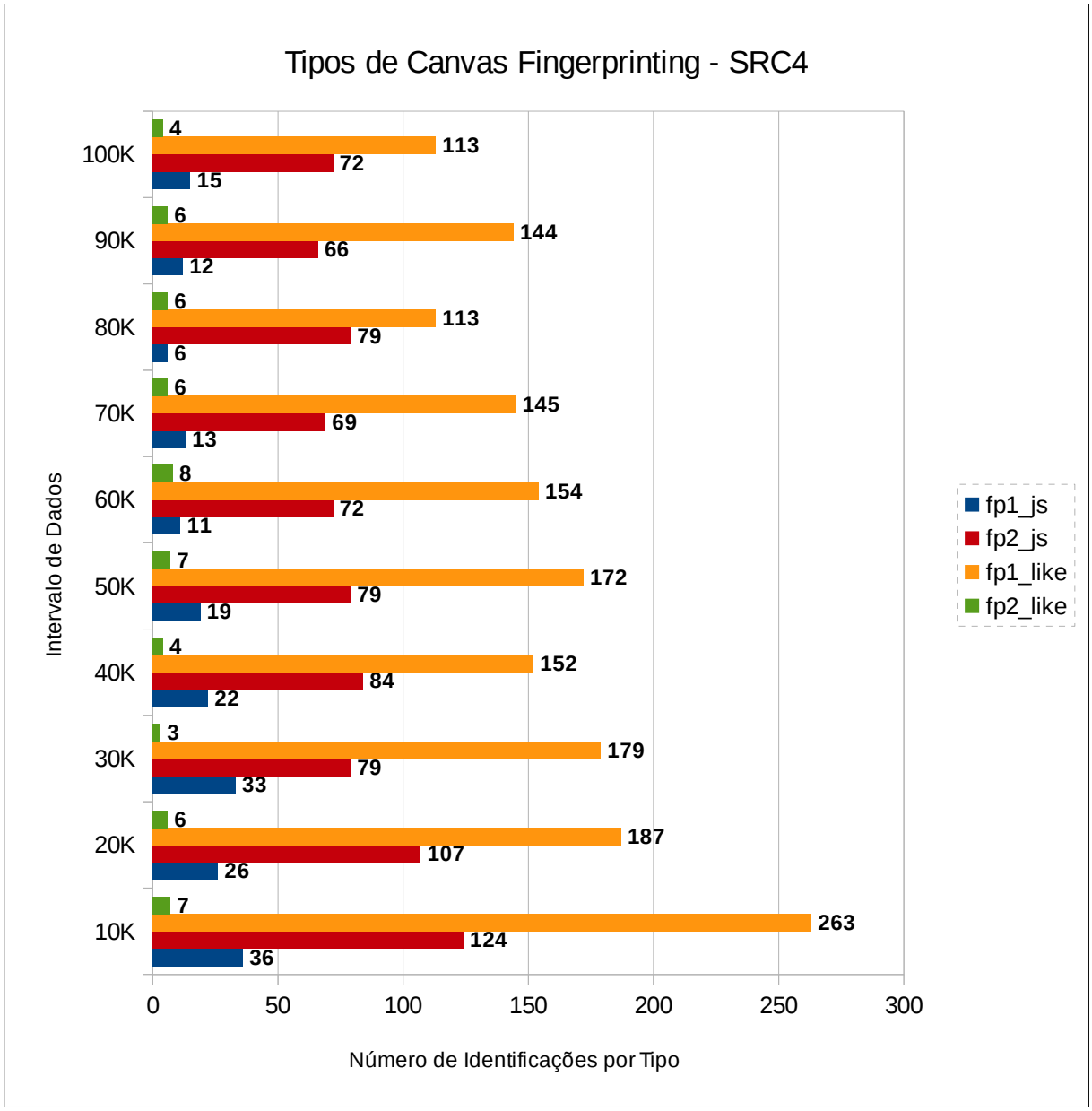

Figura 4.7 - Tipos de Canvas Fingerprinting identificados em SRC2.

Complementando os resultados visualizados na Figura 4.7, amostras de parte das imagens de canvas gerados utilizando esses tipos detectados são mostrados na Figura 4.8.

Na Figura 4.8 observe as características das imagens originalmente geradas pelos tipos fp1_js e fp2_js mostradas, como textos, cores e figuras geométricas. Em seguida observe algumas das variações dessas figuras mostradas nos tipos fp1_like e fp2_like. Observando as mesmas características de texto, cores e figuras geométricas, observamos que em fp1_like e fp2_like temos algumas das características das anteriores mas se diferenciam pelo texto e tamanho das formas

utilizadas.

Tipo	Imagem
fp1_js	
fp2_js	
fp1_like	
fp2_like	

Figura 4.8 - Exemplos de imagens dos tipos de Canvas Fingerprinting encontrados

A partir daí, são apresentados os resultados mais detalhados na Figura 4.9. De forma diferente, estes dados são os mesmos encontrados na Figura 4.7, mas apresentados numa tabela e com suas totalizações para uma melhor análise.

	Domínios em SRC4 - Fingerprinting				
Intervalo	**fp1_js**	**fp2_js**	**fp1_like**	**fp2_like**	**Totais**
10K	36	124	263	7	**430**
20K	26	107	187	6	**326**
30K	33	79	179	3	**294**
40K	22	84	152	4	**262**
50K	19	79	172	7	**277**
60K	11	72	154	8	**245**
70K	13	69	145	6	**233**
80K	6	79	113	6	**204**
90K	12	66	144	6	**228**
100K	15	72	113	4	**204**
Totais	**193**	**831**	**1622**	**57**	

Figura 4.9 - Tipos de Canvas Fingerprinting identificados em SRC2 e seus totais.

Então, levando em consideração que a nova proposta da extensão detectou 2.924 sites que utilizam Canvas Fingerprinting, tivemos 193 sites utilizando o tipo

fp1_js, 831 sites utilizando o tipo fp2_js, 1.622 sites utilizando o tipo fp1_like (tipo com maior número) e 57 sites utilizando o tipo fp2_like. Somando, temos 2.703 sites que utilizam a biblioteca FingerprintJS ou FingerprintJS2 ou alguma variação dessas, para gerar um Fingerprint do dispositivo/usuário. Assim, apesar de termos 7,56% das detecções sem enquadramento em nenhum dos tipos adotados neste trabalho, o que nos chama atenção é que temos 92,44% das detecções de Canvas Fingerprinting utilizando essas bibliotecas.

Ainda, na última coluna da Figura 4.9, temos os totais de detecções por faixa, dos sites mais acessados aos menos acessados segundo o ranque Alexa. Em média, quanto mais alto no ranque o grupo estiver, mais detecções serão encontradas nesse grupo, ou seja, quanto mais alto o grupo estiver no ranque mais sites são encontrados que utilizam mecanismos para identificar o usuário.

Por fim, observe ainda que na Figura 4.9, o primeiro intervalo (10K), teve 430 detecções, ou seja, 14,71% das detecções estão nessa faixa. A segunda faixa (20K) contempla 11,15% das detecções e a terceira faixa (30K) engloba 10,05% das detecções. Assim, essas três primeiras faixas, ou seja, os 30.000 sites mais acessados segundo o ranque Alexa, corresponde a 35,91% das detecções de Canvas Fingerprinting encontradas e que englobam mais de um terço do total de detecções.

5 CONCLUSÃO

Neste trabalho identificamos que a extensão Canvas Blocker, compatível com o navegador Firefox, na sua essência, bloqueia algumas funcionalidades da API de Canvas do HTML5. Aprimoramos esta extensão focando na detecção especificamente da utilização do mecanismos de Canvas Fingerprinting. Com isso, conseguimos disponibilizar uma extensão para Firefox que detecta e avisa ao usuário quando a página, que está sendo acessada, coletou dados e fez sua identificação utilizando o mecanismo de Canvas Fingerprinting.

Assim, a extensão da forma original bloqueia a utilização do mecanismo de Canvas. Por isso houve tantas detecções (14.176) sobre os 100 mil sites visitados, ou seja, 14,18% dos sites visitados. E como ela bloqueia a utilização do Canvas, bloqueia também, desta forma, quando ele for utilizado para gerar o Fingerprint ou o Canvas Fingerprinting.

Já, a nova proposta para a extensão foca especificamente na detecção de Canvas Fingerprinting, por isso houve uma redução grande no número de detecções que passou a ser de 2.924 (2,92%) dos 100 mil sites visitados.

Para esse aumento de eficiência, os apontamentos feitos por Acar et al. (2014) foram relevantes. Somando-se a estes apontamentos, foi constatada a grande utilização do WordPress e de Emojis pelos sites na atualidade. Desta forma, a proposta apresentada por este trabalho de identificar a utilização do mecanismos de Canvas para criar um Emoji (dentro da função `fillText)` e não criar um Canvas Fingerprinting, foi fundamental. Esta proposta acrescentou uma nova variável que deve ser considerada nos estudos de identificação de Canvas Fingerprinting na atualidade e que, quando considerada, foi essencial para reduzir os falsos positivos nessas detecções.

Outro ponto que merece destaque é a comparação com os resultados obtidos por Acar et al. (2014) que encontrou 5,5% dos sites utilizando Canvas Fingerprinting em 2014. Nessa pesquisa encontramos 2,92% dos 100 mil sites

utilizando Canvas Fingerprinting. Houve uma redução, mas a quantidade ainda é significativa, principalmente se considerarmos os 30 mil primeiros sites do ranque da Alexa onde o número de detecções passa para 1.050, ou seja, 3,5% dos 30 mil sites utilizam um mecanismo para gerar Canvas Fingerprinting dos usuários que visitam seus sites.

Chama a atenção também o fato de que, dentre as detecções, 92,44% dos sites que utilizam Canvas Fingerprinting utilizam a biblioteca FingerprintJS, FingerprintJS2 ou uma variação dessas. Isso revela quase um monopólio da utilização dessas bibliotecas para gerar o Fingerprint utilizado para identificação do dispositivo/usuário.

Contudo, apesar de ser apresentado um mecanismo de detecção de Canvas Fingerprinting, a geração de um Fingerprint não é ilegal ou contra a moral. O que questiona-se é a execução da geração desse identificador sem o consentimento, ou mesmo o conhecimento, prévio do usuário. Por isso a extensão auxilia o usuário nesse sentido, de avisar quanto à utilização do mecanismo de Canvas para gerar um Canvas Fingerprinting para a identificação do dispositivo/usuário.

Então, mesmo sendo uma extensão facilmente detectável no navegador, essa mostrou-se uma solução rápida e eficiente, já que não depende de uma alteração interna no navegador. E, com a nova proposta apresentada, a eficiência na detecção foi comprovada, com a exclusão de vários falsos positivos e melhorando a detecção focada em Canvas Fingerprinting.

5.1 Trabalhos Futuros

Pensando na própria extensão Canvas Blocker, na parte do seu código onde através de uma sequência randômica é modificada a imagem de Canvas original e criada uma imagem de Canvas “fake”. Poder-se-ia estudar formas de melhorar ou mediar a eficiência dessa sequência randômica quando utilizada para gerar imagens únicas a cada vez, dificultando ainda mais a identificação do usuário/dispositivo.

Como extensão, o identificador de Canvas é facilmente identificado durante a navegação na Internet. Pensando em dificultar essa identificação, pode-se pensar

em fazer uma sugestão de modificação nos fontes do navegador, podendo ser até no nível privado, que faça essa identificação da utilização do mecanismo de Canvas quando utilizado para gerar um Canvas Fingerprinting.

Outro trabalho um pouco mais elaborado pode ser pensado em criar uma extensão que unisse as duas técnicas de detecção. Integrar a identificação da geração de fingerprint não autorizado criado da forma tradicional (pegando apenas informações do navegador e do sistema operacional) e detecção quando se utiliza o mecanismo de Canvas Fingerprinting (fingerprint em cima de uma imagem de Canvas).

Pensando em Canvas Fingerprinting, pode-se pensar em um estudo para medir a eficiência das bibliotecas FingerprintJS e FingerprintJS2 na geração do Fingerprint que é utilizado para a identificação do usuário/dispositivo.

A título de conhecimento, pode-se pensar em pesquisar, dentre os sites que utilizam Fingerprinting, quais os que utilizam apenas Canvas Fingerprinting, pois a identificação única, como percebido com esta pesquisa, utiliza a imagem e mais algumas informações do navegador para gerar o identificador. Esta pesquisa buscaria verificar se tem algum site que utiliza apenas essa imagem para gerar um identificador único.

Ainda nessa linha, tomando por base a exigência legal, pode-se fazer um levantamento de quantos sites avisam e pedem a autorização do usuário para gerar esse Fingerprinting, para então, somente com essa autorização, guardar e processar informações do mesmo.

REFERÊNCIAS

ACAR, Gunes et al. The Web Never Forgets. **Proceedings Of The 2014 Acm Sigsac Conference On Computer And Communications Security - Ccs '14,** [s.l.], p.674-689, 2014. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/2660267.2660347.

ARNOLD, René; HILLEBRAND, Annette; WALDBURGER, Martin. **Personal Data and Privacy. Wik Consult**. Bad Honnef, 26 May 2015. Acessado em Outubro de 2015 e disponível em: <http://stakeholders.ofcom.org.uk/binaries/internet/personal-data-and-privacy/Personal_Data_and_Privacy.pdf>.

BODA, Károly et al. User Tracking on the Web via Cross-Browser Fingerprinting. **Information Security Technology For Applications,** [s.l.], p.31-46, 2012. Springer Nature. http://dx.doi.org/10.1007/978-3-642-29615-4_4.

BORGESIUS, Frederik J. Zuiderveen. **Improving Privacy Protection in the Area of Behavioural Targeting**. PhD Thesis, University of Amsterdam, 2014. Acessado em Outubro de 2015 e disponível em: <http://dare.uva.nl/record/1/434236>.

BORGESIUS, Frederik J. Zuiderveen. Personal data processing for behavioural targetting: which legal basis? **International Data Privacy Law**. Oxford Journals, June 23, 2015. Acessado em Outubro de 2015 e disponível em: <http://idpl.oxfordjournals.org/content/early/2015/06/23/idpl.ipv011.full.pdf+html>.

DHAWAN, Mohan; KREIBICH, Christian; WEAVER, Nicholas. **The Priv3 Firefox Extension**. Site web: 2011. Acessado em Setembro de 2016 e disponível em: <http://priv3.icsi.berkeley.edu/>.

ECKERSLEY, Peter. How Unique Is Your Web Browser? **Privacy Enhancing Technologies,** [s.l.], p.1-18, 2010. Springer Nature. http://dx.doi.org/10.1007/978-3-642-14527-8_1.

FIFIELD, David; EGELMAN, Serge. Fingerprinting Web Users Through Font Metrics. **Financial Cryptography And Data Security,** [s.l.], p.107-124, 2015. Springer Nature. http://dx.doi.org/10.1007/978-3-662-47854-7_7.

KHADEMI, Amin Faiz; ZULKERNINE, Mohammad; WELDEMARIAM, Komminist. An Empirical Evaluation of Web-Based Fingerprinting. **Ieee Software,** [s.l.], v. 32, n. 4, p.46-52, jul. 2015. Institute of Electrical and Electronics Engineers (IEEE).

http://dx.doi.org/10.1109/ms.2015.77.

LEI. **Constituição da República Federativa do Brasil de 1988**. Acessada em Outubro de 2015 e disponível em: <http://www.planalto.gov.br/ccivil_03/Constituicao/ConstituicaoCompilado.htm>.

LEI. **Marco Civil da Internet**. Lei nº 12.965, de 23 de Abril de 2014. Acessada em Outubro de 2015 e disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2014/lei/l12965.htm>.

MOWERY, Keaton; SHACHAM, Hovav. Pixel Perfect: Fingerprinting Canvas in HTML5. **Workshop Web 2.0 Security & Privacy 2012**. Organizado por IEEE Computer Society, Maio de 2012. Acessado em Julho de 2015 e disponível em: <http://w2spconf.com/2012/papers/w2sp12-final4.pdf>.

NIKIFORAKIS, Nick; JOOSEN, Wouter; LIVSHITS, Benjamin. PriVaricator. **Proceedings Of The 24th International Conference On World Wide Web - Www '15,** [s.l.], p.820-830, 2015. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/2736277.2741090.

ROESNER, Franziska; KOHNO, Tadoyoshi; WETHERALL, David. Detecting and defending against thirdpart tracking on the web. In **Proc. 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)**, pages 155–168. USENIX, 2012.

SARAIVA, Adriana Rodrigues et al. Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas. Minicurso do **XIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, SBSeg 2014**. Acessado em Outubro de 2015 e disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbseg/2014/0035.pdf>.

SITE ALEXA. **Alexa Top Websites**. Acessado em Fevereiro de 2016 e disponível em: <http://www.alexa.com/topsites>.

SITE CANVAS BLOCKER. **Add-ons CanvasBlocker Website**. Acessado em Junho de 2016 e disponível em em: <https://addons.mozilla.org/en-US/firefox/addon/canvasblocker/>.

SITE RANDOM USER AGENT. **Randomly generates User-Agent strings based on actual browser market share and usage statistics**. Programa com última versão lançada em 2014. Acessado em Setembrro de 2016 e disponível em: <https://github.com/jmealo/random-ua.js>.

SITE SEO BOOK. **SEO Book website**. Acessado em Fevereiro de 2016 e disponível em: <http://www.seobook.com/download-alexa-top-1-000-000-websites-free>.

SITE TOR BROWSER. **Tor Browser**. Site oficial. Acessado em Setembrro de 2016 e disponível em: <https://www.torproject.org/projects/torbrowser.html.en>.

TORRES, Christof Ferreira; JONKER, Hugo L.; MAUW, Sjouke. FP-Block: usable web privacy by controlling browser fingerprinting. In **Proc. 20th European Symposium on Research in Computer Security (ESORICS'15)**. Springer, LNCS,

2015.

WHITMAN, Michael E.; MATTORD, Herbert J. **Principles of Information Security**. Fourth Edition. Course Technology, Cengage Learning: Boston, 2012.

APÊNDICE A – Fontes da Extensão Canvas Blocker Modificados

A extensão modificada e os fontes da mesma podem ser baixados em um dos seguintes endereços:

- https://github.com/ejrstello/CanvasBlocker/tree/0.3.6-jRelease
- https://goo.gl/v1OKOD
- https://drive.google.com/open?id=0B2S2fnNEkBigeDU3d0NZeFBWSDA