TARP Fingerprinting: Um Mecanismo de Browser Fingerprinting Baseado em HTML5 Resistente a Contramedidas

Pablo Ximenes[1,3], Márcio Correia[3], Patrícia Mello[2], Fernando Carvalho[3], Miguel Franklin[3], Rossana Andrade[3]

[1]Governo do Estado do Ceará – Empresa de Tecnologia da Informação do Ceará (ETICE) – Fortaleza – CE – Brasil

[2]Instituto Atlântico – Fortaleza – CE – Brasil

[3]Universidade Federal do Ceará (UFC) – Mestrado e Doutorado em Ciência da Computação (MDCC) – Fortaleza – CE – Brasil

pablo@ximen.es, marcio@ufc.br, patriciatmello@gmail.com, fcarvalho@ufc.br,miguel@lia.ufc.br,rossana@ufc.br

Resumo. *Este artigo apresenta um novo mecanismo de browser fingerprinting baseado em HTML5 que é resistente a contramedidas. Mais especificamente, apresentamos o mecanismo TARP Fingerprinting †, que permite a geração de uma assinatura identificadora de um navegador de internet (browser) através do uso do elemento <canvas> presente na API HTML5. Diferentemente do modelo tradicional de Canvas Fingerprinting, o mecanismo apresentado é resistente às técnicas usadas como contramedidas ao Canvas Fingerprinting, especialmente àquelas que bloqueiam a técnica adulterando o comportamento normal do elemento <canvas>. O mecanismo proposto atinge esse objetivo através da modulação da entropia das assinaturas geradas pelo elemento <canvas>, que passam a funcionar também como uma espécie de "marca d'água" ou "lacre de garantia". Ademais, apresentamos um dos dois únicos estudos existentes atualmente que medem a entropia da técnica de Canvas Fingerprinting em larga escala (>64.000 amostras) com resultados de entropia de Shannon expressivos.*

1. Introdução

O mecanismo conhecido como *cookies* foi inventado por Lou Montulli em 1994 e tinha o propósito original de rastrear as atividades de um usuário em um website [PEN2000]. Cookies são pequenas *strings* de texto recebidas pelo navegador ao fazer uma conexão HTTP e associadas ao sítio web visitado, de forma automática e com mínima participação do usuário. Assim, ao retornar ao sítio web visitado anteriormente, o navegador automaticamente envia, junto com sua requisição, a *string* do cookie gravada outrora. Essa string serve para identificar o navegador web em sucessivas requisições e permite que todas essas requisições sejam associadas umas às outras em um verdadeiro sistema de rastreamento HTTP.

Contudo, o modelo de cookies possui uma fragilidade gritante: o usuário sempre poderá apagar o cookie (ou mesmo bloqueá-lo). Essa deficiência permite a existência de diversas ferramentas e condutas de usuário que inibem o rastreamento alcançado pelo uso de cookies. Assim, embora ainda amplamente utilizados, os cookies estão perdendo gradualmente sua eficiência. Seja por conta de condutas mais astutas de usuários preocupados com a privacidade, seja pelo uso de ferramentas que diminuem a capacidade

 ©2016 SBC — Soc. Bras. de Computação

de rastreamento oferecida pelos cookies. Isso se torna um problema em especial para o segmento de publicidade e propaganda online. Esta indústria, para poder entregar anúncios com contexto de real interesse a seus usuários, utiliza técnicas que dependem enormemente do rastreamento trazido pela técnica. Inviabilizar esse rastreamento pode significar anúncios menos eficazes. Outro segmento que teria muito a perder com a inviabilização dos cookies é o segmento de proteção à fraude (em especial o de proteção à fraude bancária) que depende do rastreamento de usuários para identificar comportamentos fraudulentos [NIK2013].

Por conta disso, outras formas de rastreamento de clientes WEB se fizeram necessárias. Estas alternativas necessariamente tiveram que se basear em elementos que identifiquem o browser, mas que não pudessem ser facilmente destruídos pelo usuário, como em uma espécie de "impressão digital" (ou fingerprinting) do navegador [ACA2014]. Dessa forma, diversas técnicas de browser fingerprinting foram propostas [UPA2015]. Em especial, o Canvas Fingerprinting [MOW2012] é destas técnicas a que mais obteve sucesso [ACA2014]. Ele é baseado na coleta de informações gráficas de renderização do HTML5 presente no navegador para gerar uma assinatura.

Contudo, o Canvas Fingerprinting, bem como as demais técnicas de Browser Fingerprinting propostas, depende de que o código gerador da assinatura seja executado, em grande parte, no lado do cliente, onde não existem controles de segurança que garantam que este código não foi adulterado. Isso significa dizer que o comportamento de um determinado navegador pode ser alterado de forma a forjar uma assinatura. Isso tem se tornado cada vez mais simples com a existência de navegadores que permitem a instalação de plug-ins como o FireFox e o Google Chrome [WAN2012][LIU2012]. De fato, já existem inclusive alguns plug-ins no mercado que prometem proteção contra Canvas Fingerprinting [APP2016][KKA2016]. Eles simplesmente alteram a resposta do código de coleta da assinatura para enviar dados forjados ou simplesmente lixo.

Para contornar esse problema, propomos uma variação do Canvas Fingerprinting resistente a contramedidas. Nossa técnica, chamada de *TARP Fingerprinting*[†], se vale da modulação da entropia presente na geração de imagens na técnica do Canvas Fingerprinting. O mecanismo proposto explora a incapacidade de diferenciar imagens com alta entropia de imagens com baixa entropia por parte das técnicas de contramedida ao Canvas Fingerprinting. Ao invés de usar o *hash* de uma única imagem, como no Canvas Fingerprinting tradicional, nosso mecanismo usa o conteúdo de múltiplas imagens, variando a entropia entre elas, de forma a conseguir identificar uma resposta que tente forjar a assinatura. [1]

Este artigo está organizado da seguinte maneira. A seção 2 trata de uma discussão das técnicas de fingerprinting de browser. A seção 3 fala especificamente do Canvas Fingerprinting. A seção 4 discute as contramedidas existentes contra o Canvas Fingerprinting. A seção 5 detalha nossa proposta. A seção 6 fala dos resultados obtidos, demonstrando protótipo e experimentos realizados. Por fim, a seção 7 tece algumas conclusões.

[†] O nome "TARP Fingerprinting" possui uma relevância simbólica. Enquanto "*canvas*" é a tradução inglesa de "lona", a palavra "*tarp*" (abreviação de *tarpaulin*) é a expressão inglesa usada para designar a lona que foi tratada quimicamente para se tornar à prova d'água.

 ©2016 SBC — Soc. Bras. de Computação

2. Browser Fingerprinting

De acordo com [UPA2015], as técnicas de browser fingerprinting podem ser categorizadas em 4 grupos principais. São eles: o Browser Specific Fingerprinting; o Cross Browser Fingerprinting; o Javascript Engine Fingerprinting; e o Canvas Fingerprinting.

Ao longo desta seção, detalharemos cada uma destas categorias, à exceção do Canvas Fingerprinting, que será tratado de forma exclusiva na seção seguinte, dada a sua importância basal para o modelo proposto.

2.1. Browser Specific Fingerprinting

Trata-se da primeira leva de técnicas usadas para identificar navegadores web. Ela é baseada na coleta de strings semiestatizas presentes de forma específica nos diversos navegadores. Ela faz isso através do uso de funcionalidades específicas do Browser como o Java e o Flash. Exemplos destas strings são a resolução de tela, as fontes instaladas, plug-ins instalados, a string User-Agent, dentre outros. A exemplo desta categoria, podemos citar o projeto Panopticlick [ECK2010] que foi o precursor das demais técnicas de Browser Fingerprinting aqui discutidas [ACA2014]. Nele foi demonstrado que navegadores com Flash ou Java instalados poderiam ser distinguidos dos demais de forma individual mesmo sem o uso de cookies [ECK2010].

2.2. Cross Browser Fingerprinting

Essa categoria tenta se valer de técnicas que não sejam exclusivas de um navegador específico. Por exemplo, uma subcategoria dela é o Fingerprinting de Acelerômetro. Neste exemplo, os erros de calibração do acelerômetro são utilizados como estratégia para identificar individualmente o navegador [UPA2015]. Em contraste com a categoria anterior, a Cross Browser Fingerprinting não depende de um plug-in ou nada específico do browser, que pode ou não estar instalado, para obter os dados geradores da assinatura. Poderia, por exemplo, obter a mesma lista de fontes instaladas como na técnica de Browser Specific, mas ao invés de usar o Flash ou Java, usaria o Javascript, que é uma funcionalidade presente em todos os navegadores. Essa característica de usar estratégias que funcionem em qualquer navegador é o que define a categoria.

2.3. Javascript Engine Fingerprinting

Esta categoria se vale de testes de conformidade aplicados ao Java Script para obter assinaturas. Basicamente ela realiza uma série de testes de conformidade, se valendo muitas vezes de uma suíte específica de testes para isso. Cada navegador irá "passar" em alguns testes e falhar em outros. Essa relação de testes aprovados e testes reprovados comporá uma assinatura que indicará, na maioria dos casos, o tipo e versão do navegador, de forma muito mais confiável que a análise da string "USER-AGENT" [UPA2015]. Exemplo desta categoria é o trabalho realizado em [MUL2013]. Nele, os autores realizam uma série de testes de conformidade Javascript através da ferramenta Sputnik [SPU2016] de forma a gerar uma assinatura.

3. Canvas Fingerprinting

O Canvas Fingerprinting é uma técnica relativamente recente, proposta em 2012 por Mowery e Shacham [MOW2012]. É a forma de Browser Fingerprinting mais usada na

 ©2016 SBC — Soc. Bras. de Computação

Internet atualmente [ACA2014]. Basicamente, a estratégia é desenhar uma imagem (com texto e cenas WebGL) usando comandos gráficos do HTML5 através da *tag* <canvas> para, posteriormente, fazer a captura da imagem desenhada. A imagem capturada é usada para construir a assinatura, já que suas nuances a podem deixar exclusiva para cada navegador. Essa assinatura é, em seguida, remetida ao servidor que realiza o rastreamento. Nesta seção discutiremos em detalhes essa técnica.

3.1. Detalhes de Implementação

O Canvas Fingerprinting é implementado através do uso do elemento <canvas> presente na API do HTML5. Na prática, essa técnica é aplicada através de um código em Javascript que é executado no navegador ao visitar um site. O código abaixo demonstra a simplicidade de implementação desta técnica em Javascript:

```
<script type="text/javascript"> var canvas = document.getElementById("drawing"); var context = canvas.getContext("2d"); context.font = "18pt Arial"; context.textBaseline = "top"; context.fillText("Hello, user.", 2, 2); </script>
```

Esse código é um exemplo de renderização gráfica de um texto (no caso a palavra inglesa "drawing"). O resultado é uma imagem contendo o referido texto encapsulada em um objeto <canvas> que não precisa ser necessariamente exibida na tela do usuário. O objeto <canvas>, por sua vez, possui o método toDataURL(). Quando executado com argumento "image/png", por exemplo, esse método retorna o conteúdo gráfico integral do canvas na forma de uma imagem PNG codifica em base64. O que ocorre é que cada navegador particular tem uma forma diferente de "renderizar" essa imagem. Conforme o gráfico torna-se mais complexo, mais única e exclusiva é essa forma. Essa complexidade pode ser trabalhada de forma a gerar imagens que sejam exclusivas para cada navegador e que não variem nele mesmo quando produzidas a partir de sites diferentes. Isso faz com que a imagem bem trabalhada, gerada por um Canvas, seja um verdadeiro identificador, uma verdadeira impressão digital.

Essa imagem, por sua vez, passa a ser tratada pelo script como elemento primordial na geração da assinatura. A proposta original do Canvas Fingerprinting sugere que essa imagem não seja remetida de volta por completo ao site que a solicitou. Em vez disso, os autores propões que um hash (MD5, por exemplo) seja gerada ainda dentro do browser em Javascript, de forma que apenas esse hash seja enviado de volta. Isso economizaria largura de banda na transmissão dos dados da assinatura.

3.2. Entropia

A forma mais comum de se medir a efetividade do Canvas fingerprinting é o cálculo da entropia de Shannon [MOW2012][LAP2016]. Ela é denotada pela fórmula: $E = -\sum_{i=1}^{n} \rho(x_i) log_2\, \rho(x_i)$. Em última análise, essa fórmula representa o número de bits de entropia gerados por um dado modelo de assinatura.

Em [MOW2012], os próprios propositores da técnica de canvas fingerprinting só foram capazes de realizar um experimento pequeno, com apenas 294 amostras, chegando a uma entropia de Shannon de menos 6 bits, o que é baixo. Dada a limitação deste experimento, os autores argumentam que esse valor subestima o potencial da técnica.

De fato, até recentemente a mensuração em larga escala da entropia da técnica de Canvas Fingerprinting ainda era um problema em aberto. Contudo, [LAP2016] apresentou um amplo estudo de várias técnicas de *fingerprinting* (incluindo também o

 ©2016 SBC — Soc. Bras. de Computação

Canvas Fingerprinting) em maio de 2016. Nele, o site *amiuniq.org* aplicava várias técnicas de fingerprinting aos seus visitantes, de forma que a coleta de assinaturas vem sendo realizada desde 2014. Todas as técnicas aplicadas tiveram sua entropia medida em um conjunto de mais de 100 mil amostras. O objetivo dos autores foi realizar um estudo nas mesmas proporções do projeto Panopticlick [ECK2010], mas incluindo novas técnicas como a de *canvas*. Como veremos a seguir, o nosso trabalho apresenta contribuição similar, apresentando também um estudo da entropia do canvas fingerprinting com grande número de amostras.

3.3. Problemas

O grande problema do Canvas Fingerprinting é ser altamente dependente do cliente na geração de assinaturas, onde pouco se faz no lado do servidor para conter contramedidas. De fato, o trabalho seminal inclusive propôs que a técnica fosse implementada através da execução de um script Javascript no browser do usuário que devolvesse ao servidor apenas o *hash* da imagem gerada. Essa estratégia, apesar de trazer benefícios no tocante a desempenho, mascara um grande problema de segurança. Ao delegar a execução do algoritmo para o cliente, não existem garantias de que o código remetido pelo servidor é o mesmo que foi executado. De fato, em uma perspectiva onde o cliente é adversário, o algoritmo certamente será adulterado de forma a torna-lo ineficaz. O Canvas Fingerprinting não é resistente a adulteração, justamente por isso. Ademais, qualquer técnica que tente ao menos identificar que houve adulteração torna-se inviável, já que a única informação recebida pelo servidor é o *hash* da imagem, um dado sumarizado que carrega pouca informação a respeito da execução do algoritmo em si.

Essa discussão não é apenas teórica, já que ferramentas de bloqueio do Canvas Fingerprinting que exploram essa fragilidade têm surgido [KKA2016][APP2016]. De fato, por conta de sua popularidade, a técnica acaba por tornar-se grande foco no desenvolvimento de contramedidas.

4. Contramedidas ao Canvas Fingerprinting

Os mecanismos de rastreamento de usuários na web geram uma preocupação de privacidade que estimulou a criação de técnicas de proteção. Da mesma forma que novas propostas de rastreamento surgem, diversas contramedidas contra essas mesmas propostas são criadas [CAR2014]. Isso se torna mais evidente no caso do Canvas Fingerprinting devido a sua popularidade.

De acordo com [KHA2014], podemos classificar as contramedidas que tentam proteger contra o Browser Fingerprinting em 3 categorias principais: as baseadas em aleatoriedade, as baseadas em filtragem e as baseadas em forja. Discutiremos como cada uma delas se aplica ao Canvas Fingerprinting no decorrer da seção.

4.1. Contramedidas Baseadas em Aleatoriedade

Este tipo de contramedida modifica aleatoriamente as propriedades do navegador web de forma a enganar os geradores de assinaturas. Como as técnicas de fingerprinting geram a assinatura com base nessas propriedades, esse tipo de contramedida faria com que as assinaturas geradas se tornassem também aleatórias, inviabilizando qualquer rastreamento.

 ©2016 SBC — Soc. Bras. de Computação

No caso do Canvas Fingerprinting, essa contramedida é acionada quando o navegador identifica a captura (para posterior cálculo de hash) do conteúdo de imagem renderizado. Isso pode ser feito através da identificação do uso do método toDataURL, responsável pela "captura" da imagem gerada. Nesse caso, ao invés de permitir que o hash da imagem original fosse enviado, o navegador substituiria o conteúdo do objeto por uma imagem aleatória. Assim, o hash calculado seria dessa imagem aleatória, gerando assinaturas diferentes para toda nova requisição. Os plug-ins CanvasBlocker para Firefox [KKA2016] e o CanvasFingerprintBlock [APP2016] para Chrome, por exemplo, podem ser configurados para fazer justamente isso.

4.2. Contramedidas Baseadas em Filtragem

Esse tipo de contramedida simplesmente desabilita a propriedade ou recurso usado para gerar a assinatura, ou a deixa funcionando, mas retornando resultados vazios. Isso impende a coleta de dados necessários à geração da assinatura. No caso do Canvas Fingerprinting, isso seria realizado através da completa desativação do elemento <canvas> do HTML5.

4.3. Contramedidas Baseadas em Forja

Esse tipo de contramedida mantém ativos os elementos ou propriedades necessárias à composição da assinatura, mas faz com que seus resultados sejam forjados. Por exemplo, ao invés de retornar a *string* identificadora do browser (user agente) real, o navegador retornaria uma versão falsa, possivelmente forjando um outro navegador.

No caso do Canvas Fingerprinting, essa contramedida novamente terá seu fluxo relacionado à identificação da captura do conteúdo de imagem renderizado. Novamente, isso pode ser feito através da identificação do uso do método toDataURL. Nesse caso, ao invés de permitir que o hash da imagem original fosse enviado, o navegador substituiria o conteúdo do objeto por uma imagem forjada, representando outro navegador ou mesmo inválida. Assim, o hash calculado seria dessa imagem forjada, gerando uma assinatura falsa, diferente da assinatura do próprio usuário. Os plug-ins CanvasBlocker (Firefox) e o CanvasFingerprintBlock (Chrome) também podem ser configurados para fazer isso.

4.3. Efeitos Colaterais das Contramedidas

Todas as formas de contramedida discutidas podem ter como efeito colateral o comprometimento das funcionalidades do navegador, já que a ferramenta que operacionaliza a contramedida não possui inteligência para distinguir o uso legítimo das propriedades do navegador que são modificadas. Contudo, no caso do Canvas Fingerprinting, esse efeito é menor. O plug-in CanvasBlocker, por exemplo, pode ser configurado para superar essa limitação através de whitelists. Dessa forma, o próprio usuário poderá "treinar" a ferramenta para que ela saiba quais são os sites que fazem uso legítimo e quais usam os elementos para rastreamento.

5. TARP Fingerprinting

O Tarp Fingerprinting funciona de forma similar ao Canvas Fingerprinting no tocante ao uso do elemento <canvas> do HTML5. Da mesma forma que seu antecessor, ele trabalha com a renderização de elementos gráficos no browser que são capturados para a geração

 ©2016 SBC — Soc. Bras. de Computação

da assinatura. Contudo, diferem em aspectos essenciais que representam uma evolução entre os modelos.

A proposta do TARP Fingerprinting é estender o Canvas Fingerprinting de forma que se torne resistente às contramedidas discutidas na seção anterior. Em termos gerais, modificamos o modelo original de forma que ele seja capaz de perceber quando alguma dessas contramedidas está sendo posta em prática. Dessa forma, ele poderá simplesmente descartar a assinatura falsa ou negar funcionalidade ao usuário que se recuse a fornecer sua assinatura. Isso é realizado mudando-se o algoritmo original para que trabalhe com múltiplos canvas (no mínimo dois) ao invés de apenas um. Parte dessas imagens servirá para compor a assinatura semelhante à maneira original, enquanto que a outra parte servirá como uma espécie de marca d'água, um verdadeiro "lacre de garantia" que, se violado, indicará que houve o uso de contramedida. Outra mudança importante é que o Tarp Fingerprinting não trabalha com *hashes* gerados pelo cliente. No lugar de acreditar no hash gerado pelo browser, o mecanismo proposto remete ao servidor todas as imagens geradas em sua integralidade.

Ao longo dessa seção discutiremos os detalhes de nossa proposta.

5.1. Substituição do modelo de Hash calculado no cliente

A diferença entre as duas técnicas começa com o fato de que o modelo Tarp não confia nos hashes gerados pelo browser como ocorre no modelo proposto em [MOW2012]. Nossa proposta exige que qualquer imagem gerada seja enviada ao servidor para inspeção em sua integralidade. Isso por si só já confere uma capacidade inicial de analisar a existência de contramedidas. Um *hash* é uma representação resumida de um conjunto de dados. Analisar dados através de seu *hash* é equivalente a analisar uma obra literária inteira somente através do título. Ou seja, teremos muito poucos elementos para verificar se houve qualquer adulteração da obra original se analisarmos somente o título. Quando temos acesso ao conteúdo completo, diversas estratégias podem ser aplicadas para aferir se houve a aplicação de alguma contramedida.

5.2. Grupos de Imagens

A segunda diferença entre os dois modelos é que o Tarp Fingerprinting passa a trabalhar com dois grupos de imagens ao invés de uma única imagem. É importante notar que cada um destes grupos pode ser representado por apenas uma imagem ou por várias imagens.

Como veremos a seguir, esses grupos serão usados para compor a assinatura e para identificar uso ou não de contramedida.

5.3. Imagens de Assinatura

O primeiro grupo é usado para compor a assinatura do browser, de forma similar ao Canvas Fingerprinting, mas com mais potencialidades. O fato de que estamos trabalhando com a imagem integral (ao invés de apenas seu *hash)* dá a liberdade à nossa proposta de aplicar qualquer transformação às imagens recebidas deste grupo para gerar a assinatura. Poderíamos, inclusive, usar todos os bits de seu conteúdo integral.

Esse grupo de imagens é modelado para possuir alta entropia, ou seja, ter características que fazer sua renderização não se repetir entre navegadores diferentes.

 ©2016 SBC — Soc. Bras. de Computação

5.4. Imagens de Marca D'água

O segundo grupo, por sua vez, gera imagens que funcionarão como uma espécie de marca d'água no sentido de lacre de garantia. Serão imagens de referência que devem se repetir com bastante frequência entre navegadores de forma que sua adulteração seja imediatamente percebida. Assim, essas imagens possuirão baixa entropia.

5.5. Processamento da Assinatura

A assinatura do browser no modelo Tarp Fingerprinting se processa da seguinte maneira.

Inicialmente as imagens dos dois grupos são geradas através do elemento <canvas> por um script Javascript executado no navegador. Essas imagens são capturadas pelo método toDataURL() e remetidas ao servidor. No servidor as imagens são analisadas.

Inicialmente, o servidor computa a assinatura do browser com base nas imagens do grupo de assinatura. Isso pode ser feita de várias formas, inclusive por meio do cálculo de *hash* de tais imagens. Como estamos fazendo isso dentro do servidor, que é um ambiente confiável de computação, não temos as mesmas limitações do Canvas Fingerprinting nesse aspecto.

A segunda análise feita pelo servidor se dá com o grupo de imagens de marca d'água. O objetivo é saber se houve alguma modificação nessas imagens. Como tais imagens foram construídas de forma que permaneçam sempre as mesmas (com todos os seus bits iguais) ou com poucas variações entre os vários navegadores diferentes (baixa entropia), uma alteração nelas pode ser facilmente percebida. O sistema coletaria todas as variações da imagem de referência e construiria padrão de referência de suas versões. Como a entropia dessas imagens é baixa, existirão pouquíssimas variações. A imagem que seja diferente das variações mais comuns, ou seja, que esteja fora do padrão de referência, será tratada como adulterada. Isso permite que o modelo proposto identifique que houve o uso de uma contramedida.

Como discutido na sessão anterior, as contramedidas funcionam através da modificação do elemento <canvas> de forma que retorne resultados forjados ou até mesmo nenhum resultado. Como tais contramedidas não são capazes de identificar quais são os elementos <canvas> que devem ser modificados, elas, em via de regra, alteram indiscriminadamente todos os elementos <canvas> presentes no script. Fazem isso com objetivo de se proteger contra o rastreamento. Ao fazer isso justamente nas imagens do grupo de marca d'água, estas contramedidas estão, indiretamente, ativando um alerta de sua existência. Ora, as imagens de marca d'água são construídas de forma que sempre permaneçam as mesmas ou com pouquíssimas variações. Receber uma imagem de marca d'água que foge completamente à norma indica que o browser foi modificado para adulterar o comportamento do elemento <canvas>, o que significa o uso de contramedidas.

Dessa forma, o modelo proposto poderá considerar a assinatura recebida como falsa e, a partir daí, requisitar outra assinatura ou mesmo simplesmente negar serviço ao navegador infrator.

5.6. Arquitetura

A figura 1 ilustra a arquitetura da proposta separando-a em navegador web e servidor.

 ©2016 SBC — Soc. Bras. de Computação

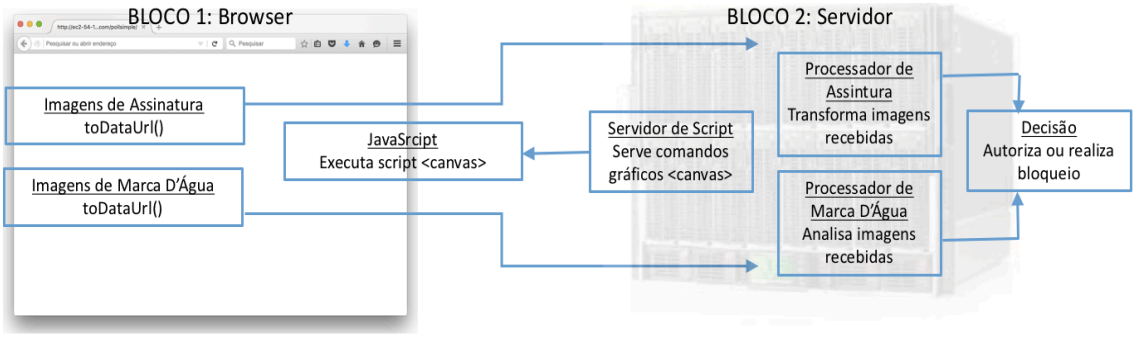

Figura 1. Arquitetura do Mecanismo Proposto

No navegador são geradas as imagens para cada um dos grupos através de script Javascript advindo do servidor. Posteriormente, cada grupo de imagens é enviado ao servidor para processamento. O processador de assinatura é responsável por transformar as imagens recebidas em uma assinatura. O processador de marca d'água é responsável por analisar as imagens recebidas para aferir se houve uso de contramedida. Os resultados dos dois processadores alimentam um módulo central de tomada de decisão que pode, inclusive, retroalimentar o mecanismo do Tarp Fingerprinting de forma que aperfeiçoe a geração de assinaturas.

5.7. Análise de Segurança

A premissa principal do mecanismo proposto é que as contramedidas aplicadas para tentar barrar a coleta de assinatura baseada no elemento <canvas> não conseguem distinguir a geração de imagens com alta entropia da geração de imagens com baixa entropia. Essa premissa, em última análise, significa que um mecanismo automático de contramedida não é capaz de obter o contexto necessário do código fonte Javascript de forma a saber qual imagem pode ou não pode ser alterada.

Contudo, para entendermos as implicações de segurança do modelo proposto, temos que definir o cenário ao qual estas se aplicam. No cenário que consideramos, temos um usuário web usando seu navegador municiado de algum plug-in de proteção contra o Canvas Fingerprinting. Nesse cenário, levando-se em conta o estado da arte, o software de contramedida será incapaz de distinguir qual <canvas> poderá ser alterado sem "disparar o alarme".

Apesar disso, devemos considerar o cenário onde o software de contramedida é customizado com ajuda do ser humano de forma estática. O programador do plug-in de proteção poderia, por exemplo, analisar uma implementação do Tarp Fingerprinting e modificar seu plug-in de forma a adicionar estaticamente a informação de qual <canvas> pode ser alterado sem acionar o mecanismo de detecção. Esse vetor de ataque pode ser facilmente debelado se programarmos o Tarp Fingerprinting para modificar os padrões de geração das imagens de marca d'água de forma dinâmica. Assim, o mecanismo proposto variaria as imagens de marca d'água de forma que não fiquem estáticas. Como a entropia de tais imagens é baixa, rapidamente o sistema se adaptaria para construir um novo padrão de referência. Isso pode ser feito através do módulo de decisão, que retroalimenta o sistema em conjunto com o módulo servidor de script. Para isso, geraríamos instâncias que incluíssem imagens já dentro do padrão de referência existente, juntamente com novas imagens diferentes, até que um padrão de referência alternativo

 ©2016 SBC — Soc. Bras. de Computação

seja construído. Assim, a contramedida nunca poderá ser adaptada para identificar imagens de marca d'água de forma estática.

6. Experimento

Para aferir o modelo proposto foi necessário avaliar, principalmente, a entropia das imagens geradas. Nosso objetivo era verificar se poderíamos gerar objetos <canvas> com imagens de alta entropia e de baixa entropia de forma simultânea para um mesmo browser. Para tanto, desenvolvemos um protótipo que implementa a proposta e aplicamos este protótipo a um conjunto de usuários web. Com isso pudemos melhor balizar nossas conclusões. Discutiremos o protótipo desenvolvido bem como o experimento realizado ao longo desta seção.

6.1. Objetivos

Modelamos um conjunto de questionamentos que funcionam como marco hipotético do experimento. Todos eles trabalham na mesma conjuntura, onde lotes distintos de instruções gráficas HTML5 com o elemento <canvas> são processadas pelo universo atual de navegadores web que, por sua vez, geram as respectivas imagens, uma para cada lote de instruções. Dessa forma, queremos responder aos seguintes questionamentos. Existe variação de entropia de assinaturas baseadas nessas imagens, quando tomadas individualmente? É possível gerar uma assinatura baseada na combinação de duas imagens que possua entropia maior do que as duas assinaturas baseadas em cada uma dessas imagens individualmente? É possível gerar ao mesmo tempo assinaturas de baixa entropia e de alta entropia com essas imagens?

6.2. Protótipo

Desenvolvemos um protótipo do mecanismo proposto para aferir a proposta com base nos objetivos apresentados. Nele, usamos três grupos distintos de instruções gráficas HTML5 do tipo <canvas>.

O primeiro grupo de instruções (grupo L) foi modelado de forma que, individualmente, produza assinaturas de baixa entropia. Ele foi construído a partir da sobreposição de duas formas gráficas simples.

O segundo grupo de instruções (grupo H) foi modelado de forma que produza assinaturas de alta entropia. Ele foi construído a partir de uma simplificação da implementação da biblioteca *fingeterprint2.js* [VAL2016]. Nesta biblioteca, são implementadas várias técnicas de Browser Fingerprinting. Escolhemos essa implementação em particular pois ela usa uma interessante diversidade de variações gráficas do elemento <canvas> HTML5 na geração da imagem. Essas variações gráficas têm o objetivo exclusivo de conferir maior entropia à assinatura gerada a partir da imagem. Contudo, o autor não dá nenhuma informação de qual a entropia gerada por suas técnicas. Assim, o que fizemos para o grupo H foi isolar somente a implementação do Canvas Fingerprinting de [VAL2016].

Finalmente, o terceiro grupo de instruções (grupo M) é uma modificação das instruções de grupo H, feita de forma a incrementar a entropia das assinaturas geradas. O que fizemos para o grupo M foi duplicar todas as técnicas de incremento de entropia usadas nas variações gráficas do grupo H, bem como inserir em codificação gráfica, na

 ©2016 SBC — Soc. Bras. de Computação

própria imagem, texto indicando a presença de certas propriedades (ex.: “canvas winding”). Um exemplo do grupo de instruções M pode ser visto na figura 2.

Figura 2. Exemplo de Imagem Gerada pelo Protótipo

Assim, o protótipo do modelo TARP gera a imagem de baixa entropia através da execução das instruções do grupo L e gera imagens de alta entropia através da execução das instruções do grupo H e do grupo M. Para diminuir o consumo de dados de armazenamento em memória, as assinaturas tratadas pelo modelo proposto são o cálculo do *hash* MD5 do arquivo binário de cada uma das imagens geradas. Dessa forma, o protótipo gera uma única assinatura de baixa entropia (que poderá ser usada como marca d’água) e duas assinaturas de alta entropia (usadas para compor um identificador distinto) por instância.

6.3. Coleta de Dados

Para avaliar com precisão a entropia gerada pelas instâncias do protótipo, desenhamos um teste de campo. Nele, instalamos o protótipo no sítio web de uma grande universidade brasileira de forma que todas as visitas gerassem uma instância do Tarp Fingerprinting da forma descrita anteriormente (imagens L, H e M).

A instalação se deu na forma da inclusão de um script em Javascript responsável por gerar as instâncias do Tarp e realizar toda a coleta de dados necessária. Todos os dados coletados estavam em consonância com os padrões de privacidade da Universidade em questão. Além dos dados tradicionais de uma transação web, foram coletadas as três imagens geradas em seu conteúdo binário integral (e não apenas o hash) para cada visita.

Nossa expectativa foi a de conseguir uma massa de dados grande o suficiente para avaliar a entropia gerada pelos grupos L, H e M com objetivo de responder os questionamentos do item 6.1.

6.4. Parâmetros de Avaliação

Para aferir os resultados obtidos usamos dois principais parâmetros de avaliação. O primeiro parâmetro é a entropia de Shannon, conforme detalhada anteriormente, já que este foi o mesmo parâmetro utilizado em [MOW2012] quando introduzida a técnica de Canvas Fingerprinting. O segundo parâmetro de avaliação é a entropia ponderada, técnica usada em [LAP2016]. A entropia ponderada é representada pela divisão da entropia de Shannon pela entropia do pior caso. A entropia do pior caso é aquela onde todas as amostras são diferentes entre si. Esse cálculo resulta em um fator que é independente do tamanho da amostra e pode ser usado para comparar amostras de tamanhos diferentes. Ademais, para melhor alinhar nossos resultados aos parâmetros de [LAP2016], também calculamos os parâmetros “assinaturas distintas” e “assinaturas únicas”. As “assinaturas distintas” são o total de assinaturas diferentes que são coletadas durante o levantamento de dados. As “assinaturas únicas” é a porção das “assinaturas distintas” que foi gerada

 ©2016 SBC — Soc. Bras. de Computação

uma única vez, nunca se repetindo. Dessa forma, usaremos os resultados de [MOW2012] e [LAP2016] como balizadores para nossos próprios resultados.

6.5. Resultados

A coleta de dados foi realizada por um período de 11 dias. As vistas recorrentes eram controladas pelo protótipo através de cookies e HTML5 *local storage* de forma a computar o visitante e não a visita. Após isolados os visitantes únicos, fomos capazes de coletar um total de 64.086 assinaturas. Uma assinatura equivale a uma instância do *Tarp Fingerprinting* gerada por um visitante único. A tabela 1 sumariza todos os resultados obtidos em conjunto com as duas últimas linhas que indicam os mesmos parâmetros, quando disponíveis, dos trabalhos [MOW2012] e [LAP2016].

Tabela 1. Sumário dos Resultados Obtidos

Instruções Canvas	total de amostras	Entropia de Shannon	Entropia Ponderada	Assinaturas Distintas	Assinaturas Únicas
Tarp L	64086	2,43	0,152	45	11
Tarp H	64086	7,68	0,481	2449	1111
Tarp LH	64086	7,68	0,481	2449	1111
Tarp M	64086	7,86	0,492	2550	1091
Tarp LM	64086	7,86	0,492	2550	1091
Tarp HM	64086	7,91	0,495	2801	1317
Tarp LHM	64086	7,91	0,495	2801	1317
[MOW2012]	**294**	**5,73**	**N/A**	**N/A**	**N/A**
[LAP2016]	**118934**	**N/A**	**0,491**	**8375**	**5533**

As linhas Tarp L, Tarp H, e Tarp M indicam os resultados obtidos pela distribuição de assinaturas (hash MD5) das imagens geradas pelos grupos de instruções gráficas HTML5 <canvas> L, H e M respectivamente, ao longo de toda a amostra. A linhas compostas (LH, LM, HM, LHM) representam a aplicação dos mesmos cálculos a composições de hashes. Em HM, por exemplo, no lugar de ver isoladamente o *hash* de H como assinatura ou o *hash* de M como assinatura, usamos as dois *hashes* concatenados como assinatura da amostra.

6.6. Análise

Com o objetivo de responder aos questionamentos de nossos objetivos do item 6.1, faremos uma breve análise dos resultados obtidos.

Iniciaremos pelo primeiro questionamento com a análise das entropias de Shannon geradas por L, H e M. Como vemos, as entropias são, respectivamente, 2,43, 7,68 e 7,86. Como cada esquema produz assinaturas de entropia diferente, oscilando para mais e para menos, temos demonstrado que é sim possível variar a entropia como questionado no primeiro quesito. Isso, em última instância, significa dizer que é possível se fazer ajustes finos nos comandos gráficos do elemento <canvas> de forma a aumentar ou diminuir a entropia da assinatura gerada.

Em relação ao segundo questionamento, podemos analisar a composição das assinaturas de H e de M. Como podemos perceber, as assinaturas de M e H são, cada uma, inferiores à composição HM, que é de 7,91. Isso significa dizer que é possível modelar gráficos distintos com o elemento <canvas> que recebem componentes entrópicos não sobrepostos. Veja que o mesmo não ocorre quando associamos L e M. A entropia de LM é exatamente a mesma da de M isoladamente, significando que os elementos entrópicos de L já estão contidos em M. Vemos, assim, que a resposta ao quesito é positiva.

 ©2016 SBC — Soc. Bras. de Computação

A respeito do terceiro questionamento, podemos analisar os extremos entrópicos que são L, com entropia de 2,43, e HM com entropia de 7,91. Vemos que L possui uma entropia bem baixa, pois é menos da metade do resultado de [MOW2012] e menos de um terço do resultado de [LAP2016]. Por sua vez, HM mostra alta entropia, sendo superior a [MOW2012] em mais de dois pontos em entropia de Shannon e superior a [LAP2016] em 0,004 pontos de entropia ponderada. Isso significa dizer que é possível gerar, pelo mesmo processo, imagens variam pouco entre as amostras e imagens que variam muito entre as amostras. A imagem com pouca variação deve se manter razoavelmente constante entre as amostras. No caso de L existem apenas 45 variações distintas entre todas as mais de 64 mil amostras. Logo, usar essa composição como marca d'água se configura viável, pois uma tentativa de burlar o esquema de *fingerprinting* modificaria também o resultado de L, gerando uma assinatura diferente das 45 levantadas pelo experimento. Isso automaticamente geraria um alerta de que determinada coleta de assinatura foi alvo de contramedida. Adicionalmente, tudo isso é feito simultaneamente à geração de outra assinatura com entropia alta, utilizável para rastreamento, segundo os melhores padrões da literatura. Dessa forma, a resposta ao questionamento 3 é positiva o que, por conseguinte, confirma a própria viabilidade da ideia por detrás de nossa proposta.

Outro ponto interessante é a similaridade de resultados quando comparamos nossos achados com os de [LAP2016]. Apesar de nosso estudo ter sido capaz de alcançar um incremento na entropia ponderada (0,004) em relação a [LAP2016], os dois trabalhos apontam para características semelhantes do Canvas Fingerprinting em termos de entropia. Contudo, se levarmos em conta que o estudo de [LAP2016] contém vícios estatísticos no sentido de favorecer usuários mais preocupados com privacidade, conforme narrado por seus autores, nossos resultados aparentam ser mais precisos. Outra vantagem de nossa proposta em relação a [LAP2016] é o tempo de levantamento de amostras. Enquanto nosso experimento foi capaz de levantar uma massa relevante de dados ao longo de dias, o experimento de [LAP2016] levou vários meses.

7. Conclusão

Neste artigo foi apresentada uma proposta de browser fingerprinting resistente a contramedidas que tentem evitar o rastreamento trazido pelo mecanismo. Trata-se de uma variação do Canvas Fingerprinting resistente a contramedidas que forjem ou falsifiquem a assinatura. Nossa técnica, chamada de TARP Fingerprinting, se vale da variação da entropia presente na geração de imagens da técnica do Canvas Fingerprinting. O mecanismo proposto explora a incapacidade de diferenciar imagens com alta entropia de imagens com baixa entropia por parte das técnicas de contramedida ao Canvas Fingerprinting. Ao invés de usar uma única imagem, como no modelo tradicional, nosso mecanismo usa o conteúdo integral de múltiplas imagens, variando a entropia, de forma a conseguir identificar um ataque que tente falsificar a assinatura. Ademais, apresentamos um dos dois únicos estudos existentes atualmente que medem a entropia da técnica de Canvas Fingerprinting em larga escala.

Referências

[NIK2013] Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., & Vigna, G. (2013, May). Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In Security and privacy (SP), 2013 IEEE symposium on (pp. 541-555). IEEE.

 ©2016 SBC — Soc. Bras. de Computação

[UPA2015] Upathilake, R., Li, Y., & Matrawy, A. (2015, July). A classification of web browser fingerprinting techniques. In New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on (pp. 1-5). IEEE.

[MOW2012] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," in Proceedings of Web 2.0 Security & Privacy. IEEE, 2012.

[LAP2016] Pierre Laperdrix, Walter Rudametkin, Benoit Baudry. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. 37th IEEE Symposium on Se- curity and Privacy (S&P 2016), May 2016, San Jose, United States.

[ECK2010] P. Eckersley, "How Unique Is Your Browser?" in 10th Privacy Enhanc- ing Technologies Symposium, 2010.

[MUL2013] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, and E. Weippl, "Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting," in Proceedings of Web 2.0 Security & Privacy. IEEE, 2013.

[SPU2016] Sputnik. Disponível em https://code.google.com/p/sputniktests/. Acessado em Maio de 2016.

[APP2016] ApPodroMe.net "CanvasFingerprintBlock: Protect your privacy. Prevent webpages from tracking you by your browser's HTML canvas fingerprint". Disponível em https://chrome.google.com/webstore/ Acessado em maio de 2016.

[KKA2016] kkapsner, "Plugin CanvasBlocker para FireFox". Disponível em https://github.com/kkapsner/CanvasBlocker/. Acessado em Maio de 2016.

[ACA2014] Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., & Diaz, C. (2014, November). The web never forgets: Persistent tracking mechanisms in the wild. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (pp. 674-689). ACM.

[LIU2012] Liu, L., Zhang, X., Yan, G., & Chen, S. (2012, February). Chrome Extensions: Threat Analysis and Countermeasures. In NDSS.

[WAN2012] Wang, J., Li, X., Liu, X., Dong, X., Wang, J., Liang, Z., & Feng, Z. (2012). An empirical study of dangerous behaviors in firefox extensions. In Information Security (pp. 188-203). Springer Berlin Heidelberg.

[PEN2000] W. Peng and J. Cisna, "Http cookies - a promising technology," Online Information Review, pp. 150–153, 2000.

[VAL2016] Valve, "fingerprintjs2: Modern & flexible browser fingerprinting library". Disponível em https://github.com/Valve/fingerprintjs2. Acessado em Maio de 2016

[KHA2014] Amin Faiz Khademi. (2014). Browser Fingerprinting: Analysis, Detection, and Prevention at Runtime. Master Thesis. Queen's University, Ontario, Canada.

[CAR2014] CARNEIRO, Brito; FEITOSA, Eduardo Luzeiro. Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas. In: Minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2014

 ©2016 SBC — Soc. Bras. de Computação