Faculdade de Engenharia da Universidade do Porto

Utilização de Reconhecimento de *Web Browsers* para Detecção de Cliques Fraudulentos

(*Using Web Browser Profiling to Detect Click Fraud*)

David Tschan Carvalho

VERSÃO FINAL

Relatório de Projecto realizado no âmbito do

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Major Telecomunicações

Orientador: Prof. Dr. Rosaldo José Fernandes Rossetti

Julho de 2008

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES 2007/2008

A Dissertação intitulada

"Using Web-Browser Profiling to Detect Click Fraud"

foi aprovada em provas realizadas 18/Julho/2008

o júri

Presidente Professor Doutor António Augusto de Sousa
Professor Associado da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Fernando Augusto Cruz e Silva Mouta
Professor Coordenador do Instituto Superior de Engenharia do Porto

Professor Doutor Rosaldo José Fernandes Rossetti
Professor Auxiliar Convidado da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - David Tschan Carvalho

Faculdade de Engenharia da Universidade do Porto

Resumo

Pagamento por clique (*Cost Per Click*) é o mais comum modelo de tarifação na publicidade *online*. Segundo este modelo, os anunciantes pagam um determinado valor por cada clique realizado nos seus anúncios. O lucro é repartido entre redes de publicidade ou motores de busca e o sítio *Web* contendo o anúncio. Apesar do seu sucesso, este modelo é vulnerável ao que é conhecido como cliques fraudulentos. Um clique fraudulento ocorre quando alguém, utilizando uma técnica manual ou automática, gera um clique falso num anúncio. Isso pode ser feito com o objectivo de esgotar o orçamento de um anunciante ou maximizar o lucro do proprietário de um sítio *Web*. Este trabalho propõe uma nova abordagem, baseada em reconhecimento de *Web browsers*, para detecção de cliques fraudulentos. Pretende-se desenvolver uma prova de conceito de detecção de cliques fraudulentos, que verifica se um *Web browser* bem conhecido foi utilizado pelo cliente para clicar num anúncio. Já que as ferramentas de clique automático procuram ocultar a sua actividade emulando *Web browsers* existentes, a verificação deve examinar a compatibilidade dos cliques com os standards que regem a *Web*, como HTTP e DOM. A prova de conceito deve correr no lado do servidor.

Abstract

Cost Per Click (CPC) is the most common business model in use in Online Advertisement. In this model, advertisers pay a certain amount for each click they get on their ads. The profit is then shared between Ad Networks/Search Engines and the website containing the Ad. Despite the success of this model, it is vulnerable to what is known as click fraud. Click fraud occurs when someone, using manual or automated techniques, generates false clicks over an Ad. This may be done to deplete the Advertiser's budget or to maximize the profit of a website owner. This work proposes a novel approach based on Web browser profiling to detect click fraud. The proof-of-concept detects click fraud by verifying if a well-known Web browser was used by the client to click an Ad. Since automated click fraud tools usually try to conceal their activity by mimicking existing Web browsers, the verification must check whether the clicks are compliant with Web standards, such as HTTP and DOM. The proof-of-concept runs on Server Side.

Índice

Lista de Figuras

Lista de Tabelas

Abreviaturas e Acrónimos

CSS	Cascading Style Sheets
CPAN	Comprehensive Perl Archive Network
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
MSIE	Microsoft Internet Explorer
ROI	Return Of Investment
SQL	Structured Query Language
TCP	Transmission Control Protocol
UAS	User-Agent String
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language

1. Introdução

1.1. Apresentação da Instituição de Estágio

A AuditMark é uma *startup* inovadora, que realiza actividades de I & D em *e-marketing*. Os objectivos da empresa são desenvolver novas soluções em duas áreas principais: auditoria de campanhas *Web* e análise de tráfego *Web*.

A AuditMark está actualmente a construir uma solução para auditoria de campanhas *Web*. Combinando inspecção e validação de tráfego *Web* com análise estatística, atinge-se uma solução tecnológica eficiente e credível, que é útil para ambos, expositor e anunciante.

A tecnologia de análise de tráfego Web está a ser desenvolvida, empregando o estado da arte das técnicas de mineração de dados, resultando em informação sobre como aumentar as receitas e melhorar o ROI (*Return Of Investiment*) das campanhas publicitárias.

1.2. Objectivos do Trabalho e Motivação

O protótipo de detecção de cliques fraudulentos por reconhecimento de browsers a desenvolver está inserido no projecto da tecnologia de auditoria de campanhas *Web*.

A publicidade na *Web* tem vindo, nos últimos anos, a consolidar-se num modelo de tarifação conhecido por CPC (*Cost Per Click*), em que os expositores cobram as campanhas publicitárias nos seus sítios *Web* em função do número de vezes que os anúncios são clicados. Os outros dois modelos em vigor são o CPM (*Cost Per Mile*) e o CPA (*Cost Per Action*). Segundo o CPM a tarifação é feita por número de visualizações do anúncio. Segundo o CPA são tarifadas as acções (preenchimento de formulários de registo, navegação em outras páginas do sítio para além da inicial, compras) realizadas por visitantes encaminhados pelo expositor para o sítio do anunciante. A predominância actual do modelo CPC explica-se por se situar num meio-termo entre os modelos CPM e CPA e, consequentemente, num ponto de equilíbrio entre os interesses do expositor (maximizar os seus lucros) e do anunciante (ter um bom retorno do seu investimento). Como em qualquer outra actividade que envolva dinheiro, este modelo atraiu uma forma de fraude designada por *Click Fraud* ou cliques fraudulentos. Não existe uma definição oficial para cliques fraudulentos, no entanto pode-se citar duas que incidem de forma clara na questão. A primeira é da Wikipédia e é a seguinte:

> "Cliques fraudulentos ocorrem no modelo CPC de publicidade *online* quando uma pessoa, *script* automatizado ou programa de computador imita um utilizador legítimo de um *Web browser* clicando num anúncio, com a finalidade de gerar uma taxação indevida por clique." [11]

A segunda definição é de clique inválido, que é o conceito preferido pelo *Google* para se referir ao fenómeno e é a seguinte:

> "Cliques gerados através de meios proibidos, e que se destinam a, artificialmente, aumentar a contagem de cliques da conta de um expositor." [12]

Estima-se que grande parte dos cliques fraudulentos sejam realizados por ferramentas de clique automático. As ferramentas de clique automático procuram imitar os cliques realizados por *browsers*. Considerando que a imitação não é perfeita, o objectivo deste trabalho é detectar cliques automáticos por contraste com cliques realizados por *Web browsers*. Para isso é necessário fazer o reconhecimento do comportamento dos *browsers* mais utilizados para posteriormente se detectar um clique não realizado por um *browser*. Reconhecer um *browser* significa ser capaz de determinar o seu nome e versão a partir de um conjunto de informações reunidas num clique por ele produzido. As informações a reunir estão relacionadas com a forma como os *browsers* aplicam e suportam os *standards* propostos pelo IETF e W3C, que regulam o funcionamento da *Web*. Os *standards* que se destacam na elaboração deste trabalho são HTTP e DOM.

1.3. Contribuições

As contribuições trazidas por este trabalho são as seguintes:

- Base de dados de perfis dos *browsers* mais populares.
- Programa de captura em tempo real de tráfego HTTP num servidor *Web*. Os dados capturados são armazenados num formato de *log* idealizado para guardar toda a informação necessária ao reconhecimento.
- Programa de reconhecimento de *browsers* por comparação dos dados dos cliques com os perfis da base de dados.
- Interface *Web* de exposição e estatística dos resultados de reconhecimento.

1.4. Resultados

Foi construído um cenário de teste, que simula um serviço de auditoria de uma campanha publicitária com expositor, anunciante, cliente e auditor. O cenário serviu para testar o desempenho do sistema de reconhecimento.

1.5. Metodologia

Um dos problemas mais críticos que os servidores *Web* enfrentam é o de identificar e validar pedidos HTTP recebidos de clientes. Os servidores *Web* são desenhados para servir *Web browsers*, mas são atacados por ferramentas automatizadas, que ao contrário dos *browsers* não enviam pedidos HTTP exactos [8]. Identificar um conjunto de pedidos e seus atributos dos principais *browsers* é uma enorme mais-valia em matéria de protecção dos servidores *Web* contra as ferramentas automatizadas. O problema tem duas camadas: 1) identificar se o pedido foi gerado por um *browser*; 2) identificar que *browser* originou o pedido.

A solução para este problema pode ser obtida a partir de diferentes métodos:

- Estudar o comportamento de vários *browsers* e tomar decisões baseadas no seu comportamento;
- Forçar os *browsers* a gerarem pedidos enviando para os servidores o produto de uma funcionalidade que as ferramentas automatizadas não são capazes de gerar;
- Forçar os compiladores dos *browsers* (de *JavaScript ou VBScript)* a gerarem pedidos para os servidores. A informação contida nesses pedidos pode ser diferente de *browser* para *browser*.

1.6. Organização do Texto

Este documento é composto por cinco capítulos. Depois da breve introdução ao enquadramento do projecto, é efectuada no segundo capítulo uma contextualização aprofundada do conhecimento reunido para a elaboração do protótipo. No terceiro capítulo é descrita a implementação do protótipo. No quarto capítulo, é descrita a estratégia de testes adoptada e apresentados os resultados. Os resultados são analisados e criticados. A conclusão é dedicada a uma introspecção global sobre o trabalho, fazendo um balanço entre os objectivos propostos e as metas atingidas. Com enfoque nos resultados serão descriminadas as qualidades do protótipo implementado e propostas melhorias para desenvolvimentos futuros.

2. Publicidade *Online*

2.1. Introdução

A maior parte das informações e serviços da *Web* chegam ao utilizador sem qualquer custo. Os expositores ou *webmasters*, por outro lado, têm despesas para proporcionarem e manterem os serviços. Para rentabilizarem os seus sítios *Web*, os expositores alugam espaços aos anunciantes. Este é o principal meio de cobrir os custos da prestação de serviços *online* actualmente. O mais famoso exemplo deste fenómeno é provavelmente o *Google*. O motor de busca acumulou uma considerável fortuna leiloando espaços junto aos resultados das pesquisas feitas pelos utilizadores [9].

A publicidade online tem vindo a atrair investimento, nos últimos anos, em detrimento de outros meios, registando um crescimento significativo e com tendência a acentuar-se. O relatório anual de publicidade *online* relativo a 2007, divulgado pelo *Interactive Advertising Bureau* (IAB) [1], afirma que entre 1995 e 2007 as receitas em publicidade *online* tiveram um ritmo de crescimento superior ao crescimento da publicidade televisiva. O IAB é uma organização de normalização e boas práticas no âmbito da publicidade *online*. A figura 2.1.1 apresenta o gráfico, que suporta a afirmação do IAB.

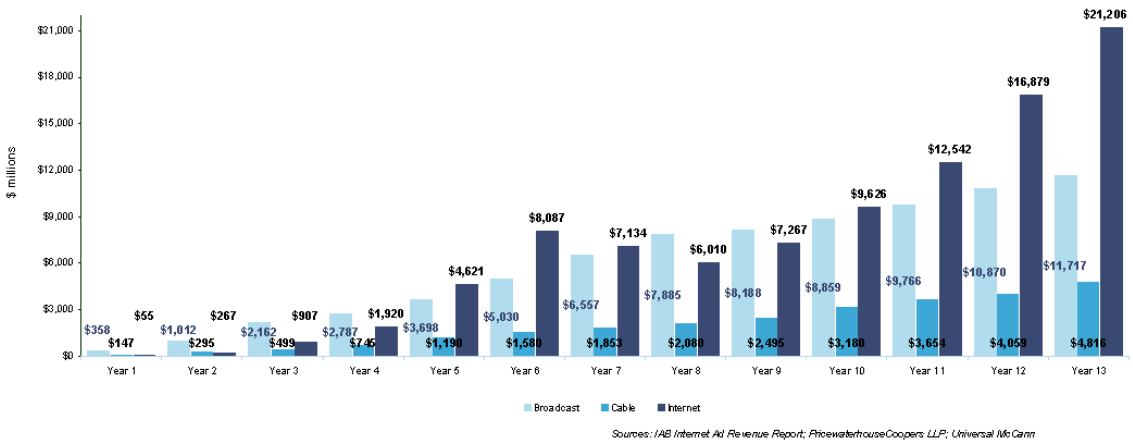

Figura 2.1.1. - Gráfico comparativo do crescimento das receitas em publicidade, da radiodifusão televisiva, da televisão por cabo e da internet. Período em análise: 1995 – 2007. Valores em milhões de dólares [1].

Os três modelos de tarifação mais comuns em publicidade online são:

- *Cost Per Mile* (CPM): O custo é determinado pelo número de vezes que o anúncio for exibido, em múltiplos de mil ocorrências.
- *Cost Per Click* (CPC): A publicidade é taxada pelo número de vezes que o anúncio for clicado. O clique no anúncio redirecciona o visitante do sítio do expositor para o sítio do anunciante.
- *Cost Per Action* (CPA): A taxação só acontece se, para além da visualização e o clique no anúncio, o visitante tomar uma determinada acção que manifeste o seu interesse pelo conteúdo do anunciante. Exemplos de acções são compras, registos (por preenchimento de formulários) ou transacções.

O relatório do IAB [1] classifica a distribuição das receitas, decorrentes da publicidade *online*, pelos modelos de tarifação, em três categorias: CPM, *Performance* (CPC ou CPA) e *Hybrid* (CPM ou CPC ou CPA). O gráfico da figura 2.1.2. apresenta a participação de cada categoria nas receitas de publicidade *online*. 51% do total de 21.206 milhões de dólares das receitas de 2007 foram taxadas segundo os modelos CPC ou CPA.

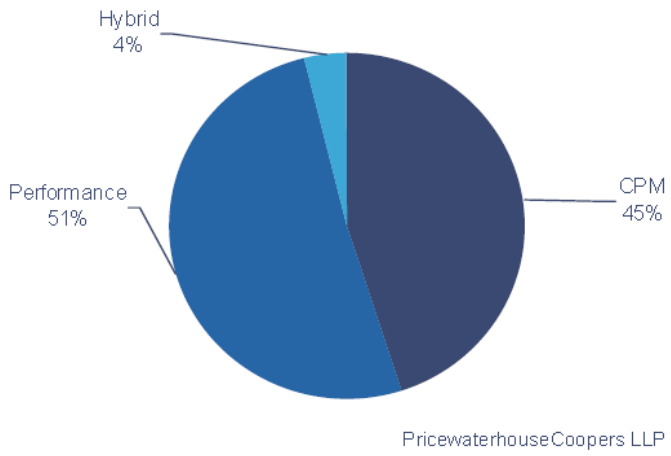

Figura 2.1.2. - Receitas em publicidade online por modelo de tarifação, no ano de 2007. Volume total: 21.206 milhões de dólares [1].

Pela sua dimensão e vulnerabilidade, o mercado de publicidade *online* tornou-se uma área muito apetecível para a prática de fraude. O fenómeno de cliques fraudulentos será exposto na próxima secção.

2.2. Cliques Fraudulentos

A publicidade online, taxada segundo o modelo CPC, é vulnerável a cliques fraudulentos. Cliques fraudulentos, em oposição a cliques regulares, não são motivados pelo interesse de

um cliente num determinado anúncio, mas sim pelo lucro que desse clique puder advir. O lucro pode ser directo, no caso de um expositor que simule cliques nos anúncios que ele próprio aloja, ou indirecto, no caso de um anunciante que simule cliques nos anúncios dos seus concorrentes com o intuito de lhes esgotar os orçamentos [7]. Da última, o praticante de cliques fraudulentos retira duas vantagens:

1. A publicidade dos seus concorrentes fica sem efeito, já que os seus anúncios não chegam a ser visitados por potenciais clientes.

2. Caso anuncie na mesma rede, o seu anúncio ganha maior proeminência, pela ausência dos anúncios dos concorrentes.

Os cliques fraudulentos podem ser realizados de forma manual ou automática. Os cliques automáticos são realizados por ferramentas que emulam a acção de um utilizador ao dar um clique num determinado anúncio. Segundo a figura 2.2.1. divulgada pela ClickForensics, empresa de auditoria na área de publicidade *online*, no primeiro trimestre de 2008, 16,3% dos cliques em anúncios foram fraudulentos [5].

Figura 2.2.1. - Evolução da taxa de cliques fraudulentos nos três últimos trimestres de 2007 e no primeiro de 2008. [5]

Os principais prejudicados da fraude são os expositores “honestos”. Pelo facto de uma parte do investimento dos anunciantes não ter retorno, por ser desperdiçada em cliques fraudulentos, o custo que estão dispostos a pagar por clique baixa. Daí serem os expositores “honestos” os principais interessados em contratar empresas de auditoria, que certifiquem a qualidade da publicidade que divulgam.

Do ponto de vista técnico é simples alguém construir um sítio *Web*, aderir a um programa de publicidade afiliado como o *Google AdSense* e manualmente dar alguns cliques nos anúncios para gerar algum lucro. Quando feito a esta escala, a quantidade de cliques ilegítimos e o seu valor é tão baixo que dificulta a sua detecção. O fenómeno só se manifesta em larga escala, quando existe recurso a ferramentas de cliques automáticos. Uma ferramenta de clique automático consiste em *software* especializado em gerar cliques em anúncios (fazer pedidos HTTP para os anunciantes) [6] e tem as seguintes variantes:

- Aplicação instalada num computador e configurada para mascarar diferentes origens por encaminhamento dos cliques via múltiplos servidores *proxy;*

- Aplicação maliciosa do tipo *trojan* instalada em centenas ou milhares de computadores sem o conhecimento do utilizador, que emulam cliques de forma concertada para enganar os mecanismos de detecção de fraude.

As técnicas de detecção de cliques fraudulentos conhecidas até à data são as seguintes:

- Correlação de cliques com mesmo endereço IP de origem.
- Monitorização de utilizadores entre sessões *Web*.
- Criação de listas negras com endereços IP e *User-Agent Strings* (UAS) de entidades ou redes suspeitas.
- Introdução de Web bugs nas páginas Web. Um Web bug, normalmente, é uma imagem invisível de 1x1 pixel. O objectivo é verificar se os objectos embebidos no documento são descarregados. Esta técnica detecta ferramentas que apenas descarreguem o documento.
- Mineração de dados e abordagens estatísticas.
- Recolha de informação única dos dispositivos físicos de acesso à rede, que permita a sua identificação.
- Reconhecimento de Web *browsers*.

A técnica de detecção por reconhecimento de Web browsers, que será implementada, requer um estudo sobre Web *browsers*.

2.3. *Web Browsers*

2.3.1. História e Mercado de *Web Browsers*

O primeiro *Web browser* gráfico popular foi lançado em 1993 pela NCSA e chamava-se *Mosaic*. Inicialmente, só corria em *Unix*, mas foi adaptado para outras plataformas como *Windows* e *Macintosh* mais tarde. O chefe da equipa *Mosaic*, Marc Andreessen, abandonou a NCSA e fundou sua própria empresa, conhecida mais tarde como *Netscape Communications Corporation*. O *Netscape Navigator* foi lançado em 1994.

Em 1995, a *Microsoft* finalmente chegou ao mercado com o *Internet Explorer* adquirido da *Spyglass, Inc*. A versão 1.0 do *Internet Explorer* não teve impacto quase nenhum, mas a partir de versão 2.0 a guerra com o *Netscape Navigator* foi iniciada. Em 1996, com a versão 3.0, o *Internet Explorer* começou a ombrear com a sua concorrência. A versão 3.0 oferecia suporte para *scripting* e foi a primeira a implementar CSS. Durante esta época, era comum entre os programadores Web exibirem um logótipo nas páginas informando “melhor visualizado

por *Netscape*" ou "melhor visualizado por *Internet Explorer*". Estes logótipos eram sintomáticos da divergência entre as normas suportadas pelos *browsers*.

Em 1998, as posições inverteram-se – o *Internet Explorer* passou à frente e o *Navigator* viu a sua quota cair drasticamente. A inclusão do *Internet Explorer* na versão base do *Windows* e melhor suporte de *standards* do W3C seriam parte da explicação para a reviravolta. Não sendo possível continuar a financiar o desenvolvimento do seu produto, a *Netscape* converteu o seu produto em código aberto, criando o *Mozilla*.

Em 2004, foi lançado o *Mozilla Firefox 1.0*. O *Firefox 2* saiu em 2006 e consolidou uma quota de mercado de aproximadamente 12%.

À margem da guerra entre *Netscape/Mozilla* e *Internet Explorer* foram surgindo outros *browsers*, como o *Opera*, o *Safari* e o *Konqueror*, cada um com o seu papel no mercado:

- O *Opera* foi lançado em 1996. É compatível com os mais importantes sistemas operativos. A versão *Opera Mini* própria para dispositivos móveis tem, actualmente, uma grande popularidade.
- A *Apple* desenvolveu o *browser Safari* para o *Mac OS*. Mais tarde, a *Apple* disponibilizou o *Safari* também para *Windows*.
- O *Konqueror* pertence à equipa do KDE e funciona na maioria dos sistemas operativos baseados em *Unix*.

Actualmente, como se pode observar pela tabela 2.3.1.1., os dois browsers mais populares continuam a ser o *MSIE* e o *Firefox*. O *MSIE* mantém-se líder de mercado com uma grande vantagem sobre o seu principal concorrente. Os *browsers MSIE*, *Firefox*, *Safari* e *Opera* representam 98,89% do mercado. Nos restantes 1,11% estão incluídos o *Konqueror* e outros.

Tabela 2.3.1.1. – Distribuição do mercado de *browsers* [2].

Browser	**Quota (%)**
MSIE	75,47
Firefox	16,98
Safari	5,82
Opera	0,62
Other	1,11

2.3.2. *Layout Engines*

A *layout engine* é a parte do browser que interpreta o código HTML e dispõe os objectos gráficos no ecrã. A layout engine carrega o *Document Object Model* (DOM), que interage com o código *JavaScript*. A mesma *layout engine* pode ser utilizada por vários *browsers*. A tabela 2.3.2.1. apresenta as *layout engines* mais populares e os *browsers* que as incorporam.

Tabela 2.3.2.1. – *Layout engines*, respectivos produtores e *browsers*.

Developer	Layout Engine	Web Browser
Microsoft Corporation	Trident	Internet Explorer
Mozilla Foundation (Currently) Netscape Corporation (Formerly)	Gecko	Firefox, Thunderbird, SeaMonkey, Sunbird, Netscape, Camino, Ephiphany, Galeon
Opera Software	Presto	Opera
Apple Inc., KDE Team, Nokia, Adobe, Google, others	WebKit	Safari, OmniWeb
Apple Inc., KDE Team, Nokia, Google, others	KHTML	Konqueror

2.3.3. Características Distinguíveis de *Web Browsers*

HTTP

O desenvolvimento do protocolo HTTP foi coordenado pelo W3C e pelo IETF culminando no RFC 2616 [3], que define o HTTP/1.1, a versão em uso actualmente. HTTP é um protocolo, de pedido/resposta entre um cliente e um servidor *Web*.

A figura 2.3.3.1. ilustra a estrutura de um pedido HTTP. A primeira linha é a linha de pedido (*Request Line*), onde é definido o método (*Method*), o Uniform Resource Locator (URL) do servidor a quem se dirige e a versão HTTP suportada pelo cliente (Version). Os parâmetros são separados por espaço (SP) e a mudança de linha é assinalada por <CR><LF> (*Carriage Return, Line Feed*). Existem oito tipos de métodos: *HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS* e *CONNECT*. Os tipos mais comuns usados pelos browsers são o GET e o POST. O GET pede uma representação da fonte especificada no URL e o POST submete informação a ser processada, que é passada no corpo do pedido. A seguir à linha de pedido estão as linhas (ou campos) do cabeçalho (*Header Lines*). As linhas do cabeçalho servem para transmitir ao servidor, o tipo de informação que o cliente suporta como resposta. Segue-se o corpo do pedido (*Message Body*), que é opcional.

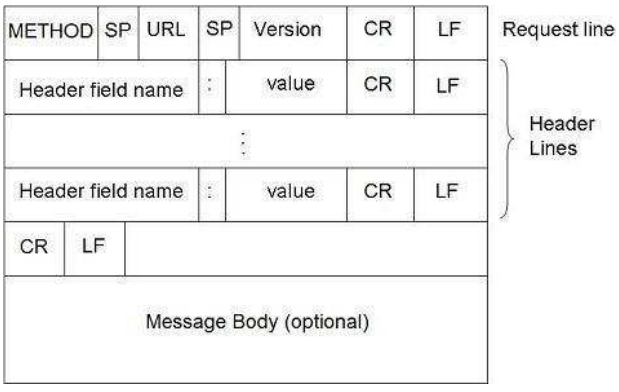

Figura 2.3.3.1. – Estrutura de um pedido HTTP.

Existem vários tipos de linhas de cabeçalho: padrão (presentes nos pedidos dos browsers mais populares), de gestão da *cache* dos browsers, de reencaminhamento por servidores *proxy*, de manutenção de sessão (para navegação em áreas que requerem autenticação) e

não normalizados específicos a certos browsers. As linhas interessantes para o tema são as padrão e as específicas a certos browsers. As linhas padrão são as seguintes:

- *Accept*: Tipos de conteúdo aceitáveis;
- *Accept-Charset*: Conjuntos de caracteres aceitáveis;
- *Accept-Encoding*: Tipos de compactação aceitáveis;
- *Accept-Language*: Idiomas aceitáveis;
- *Host*: O nome do domínio do servidor;
- *Referer*: Permite ao cliente revelar qual o URL donde foi obtido o URL do pedido actual;
- *User-Agent*: *User-Agent String* (UAS), com dados sobre o browser e sistema operativo do cliente;
- *Connection*: Especifica opções da ligação ao servidor;
- *TE*: Tipo de compactação da transferência.

As linhas específicas são a *Keep-Alive*, que é específica ao *Firefox* e a *UA-CPU*, que é específica ao MSIE.

A resposta do servidor tem uma estrutura análoga, tal como pode ser observado na figura 2.3.3.2. Na primeira linha é dado o status da resposta (*Status Line*). A linha de status apresenta a versão de HTTP suportada pelo servidor, o código e a frase de status. Existem cinco códigos e frases de status:

- 1** — Informação;
- 2** — Sucesso;
- 3** — Redireccionado;
- 4** — Erro do cliente;
- 5** — Erro do servidor.

A seguir à linha de status estão as linhas de cabeçalho (*Header Lines*) com informação relativa ao conteúdo a transmitir. No campo de dados (*Data*), é inserido o ficheiro pedido pelo cliente.

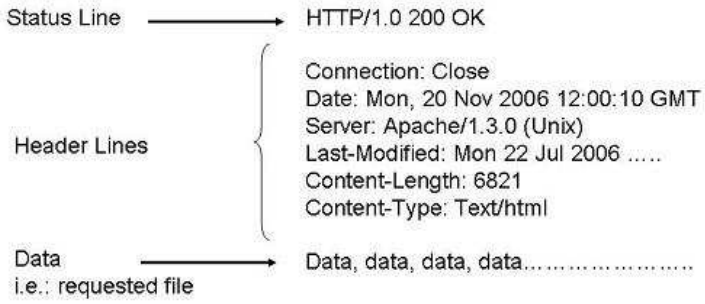

Figura 2.3.3.2. - Estrutura de uma resposta HTTP.

DOM

A especificação DOM define uma interface e uma plataforma neutras que permitem que programas e *scripts* (*JavaScript* ou *VBScript*) acedam e alterem o conteúdo, a estrutura e o estilo dos documentos, de uma forma normalizada [10]. Os objectos do DOM são carregados pela *runtime engine* do *script* do *browser*. O DOM é constituído pelos seguintes objectos:

- *Window*: É o objecto do topo da hierarquia e contém propriedades e métodos que se aplicam à janela do *browser*. Existe também um objecto *window* para cada *child window* num documento com frames.
- *Navigator*: Contém propriedades e métodos relativos ao *browser* do cliente.
- *Screen*: Contém propriedades relativas ao ecrã do cliente.
- *Document*: Contém propriedades e métodos relativos ao conteúdo do documento, tais como título, cor de fundo, hiperligações e formulários.
- *Location*: Contém propriedades e métodos relativos ao URL actual.
- *History*: Contém propriedades e métodos relativos às páginas que o cliente visitou previamente;

2.3.4. Reconhecimento de *Web Browsers*

O projecto *browserrecon* desenvolvido por Marc Ruef, um consultor suíço da área de segurança informático, é o estado da arte, pelo menos do que é conhecido publicamente, no tema de reconhecimento de *browsers*. A implementação actualmente disponível em PHP identifica o *browser* através dos seus pedidos HTTP, analisando os cabeçalhos e comparando-os com uma base de dados de “impressões digitais” de *browsers* [13].

A aplicação desenvolvida funciona como ilustrado no diagrama da figura 2.3.4.1. Um pedido HTTP enviado por um cliente é lido (*Read HTTP Headers*), iniciando-se processo de identificação. O pedido é dissecado (*Dissect Response*) para se identificarem alguns elementos específicos das “impressões digitais”. Esses elementos, um a um, são procurados na base de dados de “impressões digitais” (*Find Fingerprint*). Se ocorrer uma correspondência (um *match*), a “impressão digital” que contém o elemento correspondido é sinalizada como "identificada". Todos esses sinalizadores são contados (*Count identified Matches*) de forma ao *browserrecon* ser capaz de determinar qual a “impressão digital” com melhor taxa de correspondência. Por fim, o *browserrecon* apresenta o resultado de reconhecimento (Show computed Results).

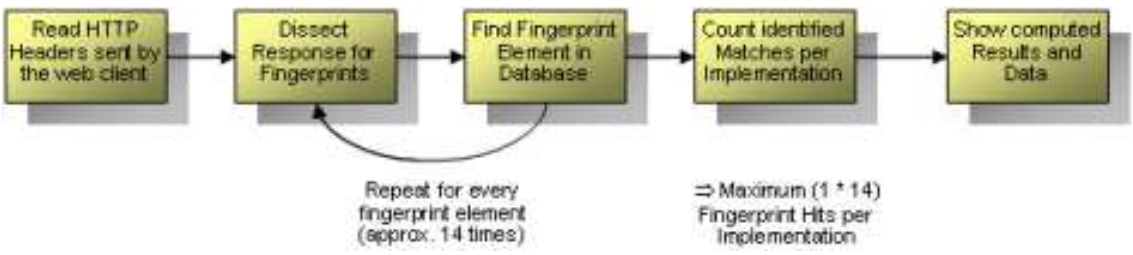

Figura 2.3.4.1. - Diagrama de blocos do projecto *browserrecon* [13].

2.4. Conclusão

O projecto *browserrecon* tem algumas limitações relacionadas com a metodologia utilizada. O sistema foi programado em PHP. O acesso ao conteúdo dos pedidos HTTP é feito através da função *getallheaders()* do PHP, que tem um formato próprio de organização dos cabeçalhos, diferente do original. Este facto restringe o total aproveitamento da informação contida nos pedidos, como por exemplo a ordem dos cabeçalhos.

Um segundo aspecto é a não consideração de outro tipo de informação para além da relacionada com HTTP. O standard DOM, por exemplo, pelas suas várias características distinguíveis poderia reforçar o grau de certeza do reconhecimento.

Por experiências feitas com alguns *browsers*, também se percebe que o *browserrecon* não faz qualquer filtragem das "impressões digitais" a incluir na base de dados, por não tomar em consideração a partilha de *layout engines* por vários *browsers*. As *layout engines* determinam o comportamento HTTP do *browser*, por isso se dois *browsers* diferentes incorporam a mesma *layout engine,* esse dois *browsers* têm o mesmo comportamento HTTP, portanto indistinguíveis desse ponto de vista. A manutenção de mais do que uma "impressão digital" para a mesma *layout engine* na base de dados, traz ambiguidade ao reconhecimento (empates nas pontuações) e redundância ao processamento.

3. Implementação do Protótipo Experimental

3.1. Introdução

A implementação do protótipo experimental tem três fases: 1) desenvolvimento do sistema de reconhecimento; 2) concepção de um serviço de auditoria com todos os elementos envolvidos; 3) elaboração de uma interface *Web* de apresentação de resultados. O sistema de reconhecimento tem duas componentes: 1) perfis dos *browsers;* 2) programa de reconhecimento.

3.2. Perfis dos *Web Browsers*

A figura 3.2.1 dá uma perspectiva gráfica da componente de criação de perfis de *browsers* (Browser Profiles) do sistema de reconhecimento. Um perfil é uma combinação única de propriedades relativas a um conjunto de características. Na figura pode-se observar um *log* com cliques provenientes de diversos tipos de *browsers*. Os *browsers* ilustrados têm como características a forma (*shape*) e a cor (color), por analogia às características HTTP e DOM. Cada tipo de *browser* tem propriedades diferentes, por vezes com mais de uma variante, como é exemplo o *browser* 1. O *browser* 1, tem duas variantes de forma, nomeadamente circular e em cruz e três variantes de cor, designadamente vermelho, verde e azul. As propriedades de cada tipo de *browser* são extraídas do *log* pelo programa BrowserProfiles, reunidas num perfil e guardadas numa base de dados (*Database*). A tabela da figura apresenta o conteúdo da base de dados depois de criados os perfis.

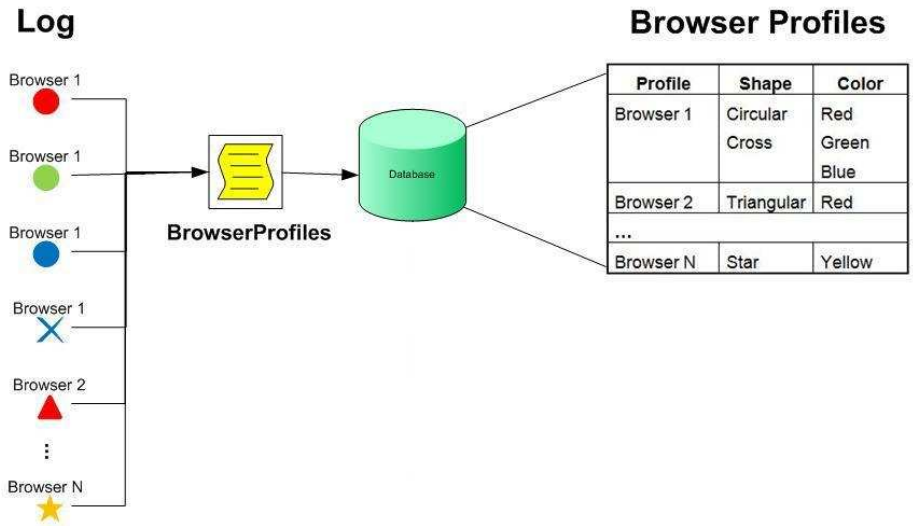

Figura 3.2.1. - Diagrama da componente de criação de perfis do sistema de reconhecimento.

Como referido acima, os perfis dos browsers têm dois subconjuntos de características, nomeadamente HTTP e DOM.

3.2.2. Subconjunto HTTP

A criação de perfis HTTP envolve dois programas, o AuditRec e o BrowserProfiles, que foram desenvolvidos durante este projecto. Perl foi a linguagem de programação escolhida, pois tem a vantagem de possuir módulos próprios para a análise de tráfego, como é exemplo o Net::Pcap disponibilizado pelo CPAN.

Programa AuditRec

O programa AuditRec tem duas versões, uma *offline* e outra *online*. Para a criação dos perfis é usada a versão *offline* que constrói logs HTTP a partir de tráfego TCP/IP. Cada entrada do *log* HTTP corresponde a um pedido. Os campos do *log* são os seguintes:

- Número do pedido HTTP;
- Endereço IP de origem;
- Endereço IP de destino;
- Porta TCP de origem;
- Porta TCP de destino;
- Todos os cabeçalhos HTTP (de acordo com o RFC 2616 [3]);

- Cabeçalhos HTTP não normalizados ou específicos de alguns browsers:

    1. Keep-Alive;
    2. X-Forwarded-For;
    3. UA-CPU;
    4. x-flash-version;
    5. Cookie;
    6. Cookie2.

- Ordem em que os cabeçalhos se encontram no pedido HTTP. Só é registada a ordem dos cabeçalhos padrão, que são os seguintes:

    1. Host;
    2. User-Agent;
    3. Accept;
    4. Accept-Language;
    5. Accept-Encoding;
    6. Accept-Charset;
    7. Keep-Alive;
    8. Connection;
    9. TE;
    10. UA-CPU;
    11. Referer.

- Nome e versão do browser (retirados da UAS).

O formato da ordem é exemplificado na figura 3.2.1.1. Cada cabeçalho da ordem tem duas coordenadas. A primeira coordenada refere-se ao cabeçalho. Cada cabeçalho tem associado um número, como se pode ler acima. A segunda coordenada é a da posição do cabeçalho no pedido e está implícita pela posição do registo de ordem.

Pedido HTTP:

1. Accept
2. Referer
3. Accept-Language
4. UA-CPU
5. Accept-Encoding
6. User-Agent
7. Host
8. Connection

Formato da ordem, tal como registado no log: 7 6 1 3 5 - - 8 - 4 2

Formato detalhado da ordem:

Host	User-Agent	Accept	Accept-Language	Accept-Encoding	Accept-Charset	Keep-Alive	Connection	TE	UA-CPU	TE
7	6	1	3	5	-	-	8	-	4	2

Figura 3.2.2.1. – Exemplo do formato da ordem dos cabeçalhos num pedido HTTP.

O nome do ficheiro de captura a ser processado é dado como parâmetro de execução do programa. A captura é processada da seguinte forma:

- Abertura do ficheiro de captura. O ficheiro contém tráfego TCP/IP bidireccional, capturado na porta 80 do servidor, que é a porta reservada ao tráfego HTTP, por omissão.
- Filtragem dos pacotes TCP/IP. Devem ficar apenas os pacotes com pedidos HTTP, tendo de se verificar ambas as seguintes condições:
    - Porta TCP de destino 80. A porta 80 é a do servidor, os pacotes com porta de destino 80, são pacotes com origem no cliente. Os pedidos são feitos pelo cliente.
    - *Flags* PSH e ACK activas.
- Para cada pacote filtrado, guardar numa *string* a seguinte informação a inserir no log:
    - Índice do pedido.
    - Do cabeçalho do pacote, os endereços IP de origem e destino e as portas TCP de origem e destino.
    - Do campo de dados do pacote, os cabeçalhos HTTP e a sua ordem.
    - Do cabeçalho *User-Agent*, o nome e versão do browser, que produziu o pedido.
- Imprimir a *string* para um ficheiro de texto.

A versão *online* do AuditRec igual à versão *offline*, com excepção dos seguintes aspectos:

- O tráfego é capturado em tempo real directamente na interface de rede do servidor;
- São acrescentados ao *log* elementos relacionados com DOM e segurança;
- É gerado um *error log*, que regista o tempo (data e hora) e estado de recepção de cada clique. Também são enumerados cabeçalhos HTTP não previstos pelo AuditRec.

Programa BrowserProfiles

O perfil HTTP de um determinado browser abrange toda a variedade de pedidos HTTP que o browser produz. A variedade dos pedidos depende dos cabeçalhos que os integram e respectivo conteúdo. A melhor forma de construir perfis é recorrendo a um log HTTP. Num log existem muitos pedidos feitos por cada browser e daí se podem extrair todas as variantes.

O BrowserProfiles tem três parâmetros de execução:

- **-i:** Nome do ficheiro de *log* HTTP a ser processado;
- **-b:** Limiar mínimo de ocorrências de um perfil para a sua admissão na base de dados;

- **-t:** Limiar mínimo de ocorrências de uma variante de característica para a sua admissão na base de dados.

O algoritmo de processamento do *log* HTTP funciona da seguinte forma:

- Para cada linha do *log* (cada pedido HTTP):
    - Ler os campos de browser, nome e versão. Se não existe registo para o browser lido, criar registo com contador inicializado a um. Se já existe, incrementar o respectivo contador.
    - Para cada campo seleccionado:
        - Ler os campos browser, nome e versão e o campo HTTP seleccionado. Se não existe registo para a combinação browser campo, criar registo com contador inicializado a um. Se já existe, incrementar o respectivo contador.
    - Inserir registos de browsers e combinações de browser campo na base de dados de perfis.

Os parâmetros **–b** e **–t** permitem a optimização dos perfis. O parâmetro **–b** define um limiar mínimo de ocorrências para os perfis e o parâmetro **–t** define um limiar mínimo de ocorrências para as variantes das características HTTP. A optimização tem o objectivo de melhorar o desempenho do AuditProfiler, que compara os cliques a reconhecer com cada perfil criado. As melhorias evidenciam-se no tempo de processamento e na precisão dos resultados. O tempo de processamento é melhorado, porque se diminui o número de comparações efectuadas pelo AuditProfiler. A precisão de resultados também é melhorada, porque se evitam ambiguidades entre perfis.

O parâmetro **–b** permite a não consideração de perfis, cuja base de conhecimento seja muito baixa (aqueles perfis que tiveram muito poucas ocorrências nas capturas efectuadas). Esses perfis correspondem a *browsers,* cuja representatividade de mercado é muito baixa. Um perfil deve ter um número mínimo de ocorrências. Caso contrário, não é garantido que todas as variantes de pedidos, que efectua, sejam abrangidas. Não considerá-los não prejudica o reconhecimento da maioria dos cliques gerados por um tráfego normal, porque têm muito baixa presença num tráfego normal.

O parâmetro **–t** permite a não consideração de variantes não válidas. Variantes não válidas constituem ruído no processo de reconhecimento. Como descrito acima, o BrowserProfiles utiliza os campos *Browser Name* e *Browser Version*, que são preenchidos pelo AuditRec versão *offline* lendo essa informação na UAS, para identificar o perfil a preencher. Pelas mais variadas razões, fraudulentas e não fraudulentas, nem sempre a UAS corresponde à aplicação utilizada pelo cliente. Isso pode ser prejudicial para a criação dos perfis. Numa captura em que um cliente, por exemplo, utilize o *Firefox 2* e o apresente como *MSIE 7*, as características do *Firefox 2* capturadas serão incluídas no perfil *MSIE 7*. A consequência é que em certos casos, um clique proveniente de um *Firefox 2* tenha pontuação da componente HTTP tão alta para o perfil *Firefox 2* como para o perfil *MSIE 7*. Casos como o exemplificado, ocorrem poucas vezes e difusamente, daí ser possível não considerá-los definindo o valor certo para **–t**.

Criação dos Perfis HTTP

É necessário um grande volume de tráfego para a criação dos perfis. O tráfego tem de contemplar todas as variantes de pedidos feitos pelos browsers, que são utilizados pela maioria dos clientes. Esse volume de tráfego foi-nos facultado por um sítio *Web*, que pela sua popularidade, reúne em pouco tempo um volume suficiente. O tráfego foi capturado pela ferramenta *tcpdump,* no servidor onde está alojado o sítio, introduzindo o seguinte comando na linha de comandos:

```
tcpdump -nnvvXSs 1514 -c [número de pacotes a capturar] -w [nome do
ficheiro de captura] port 80
```

A captura é feita na porta 80, porque é nessa que se estabelece, por omissão, o tráfego HTTP. Cada captura resulta num ficheiro de extensão *pcap* (*Packet Capture*). Foram efectuadas quatro capturas, designadamente:

1. log1pp.pcap com 501.258 pacotes;
2. log2pp.pcap com 2.000.000 pacotes;
3. log3pp.pcap com 5.000.000 pacotes;
4. log4pp.pcap com 5.000.000 pacotes;

Os ficheiros *pcap* gerados são transferidos por *Wget* para o servidor AuditSrv. O *Wget* é um programa, que permite descarregar conteúdos a partir de servidores *Web*, via HTTP ou FTP.

Obtidos os ficheiros de captura, que contêm todo o tráfego registado na porta 80 do servidor do Palco Principal, é necessário filtrá-lo e passá-lo para o nosso formato de *log* HTTP. Esse processo é feito pelo programa AuditRec versão *offline* (AuditRec-vOFF.pl). O passo seguinte é executar o programa BrowserProfiles introduzindo como entrada o *log* resultante. O BrowserProfiles extrai a informação necessária à criação dos perfis, e regista-a na base de dados. A figura 3.2.1.2 ilustra o processo que transforma os ficheiros *pcap* recolhidos em perfis.

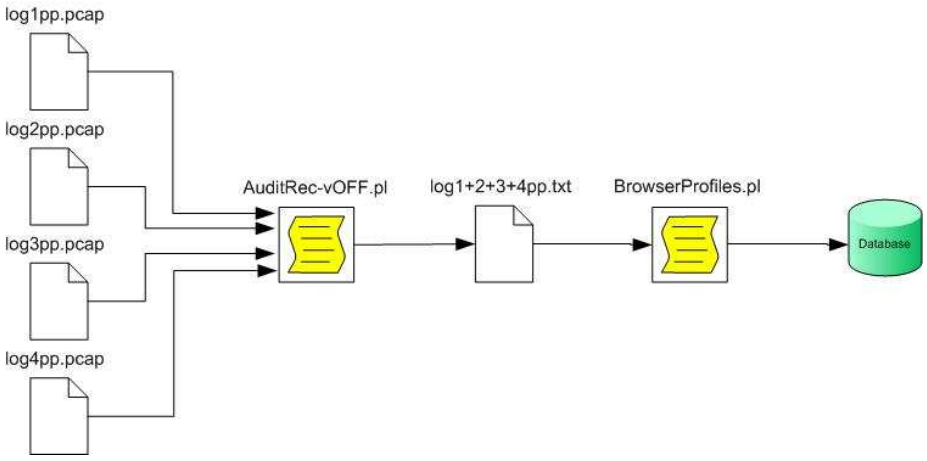

Figura 3.2.2.2. – Diagrama de criação dos perfis HTTP.

Introduzindo como ficheiros de entrada os quatro ficheiros *pcap*, obtemos como saída um ficheiro de *log* com 374.896 linhas, em que cada linha corresponde a um pedido HTTP. Antes de se executar o BrowserProfiles, tem que se escolher as características HTTP a incluir nos perfis. Os critérios que orientam a escolha das características HTTP são os seguintes: 1) grau de diferenciação na forma como são aplicadas pelos *browsers*; 2) presença não condicional nos pedidos. São excluídos, portanto, os cabeçalhos condicionais. Exemplos de cabeçalhos condicionais estão relacionados com a passagem dos pedidos por servidores *proxy* ou com gestão de *cache* ou com navegação em áreas que requerem autenticação. Observando os pedidos HTTP feitos pelos *browsers* mais populares, identificam-se as características que se inserem nos critérios estabelecidos. Para cada pedido são destacadas as suas características distinguíveis:

- *Firefox 2*:

```
GET image.jpg HTTP/1.1
Host: www.host.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; pt-BR; rv:1.8.1.12)
Gecko/20080201 Firefox/2.0.0.12
Accept: image/png,*/*;q=0.5
Accept-Language: pt-br,pt;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.referer.com
```

  - Ordem dos cabeçalhos;
  - Conteúdo do cabeçalho *Accept*;
  - No cabeçalho *Accept-Language*, “pt-br” com ‘br’ minúsculo;
  - No cabeçalho *Accept-Encoding*, formatos de compressão apenas separados por vírgula (sem espaço);
  - Inclusão do cabeçalho *Keep-Alive*;
  - No cabeçalho *Connection*, “*keep-alive*” com ‘k’ e ‘a’ minúsculos.

- *Opera 9*:

```
GET image.jpg HTTP/1.1
User-Agent: Opera/9.26 (Windows NT 5.1; U; Edition Clix; pt)
Host: www.host.com
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png,
image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: pt-PT,pt;q=0.9,en;q=0.8
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Referer: http://www.referer.com
Connection: Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
```

  - Ordem dos cabeçalhos;
  - Conteúdo do cabeçalho *Accept*;
  - No cabeçalho *Accept-Charset*, “iso” em letras minúsculas;

- No cabeçalho *Accept-Encoding*, maior variedade de formatos de compactação;
- No cabeçalho *Connection*, "TE" para além de *Keep-Alive*;
- Inclusão do cabeçalho *TE*.

MSIE 7:

```
GET image.jpg HTTP/1.1
Accept: */*
Referer: http://www.referer.com
Accept-Language: pt
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: www.host.com
Connection: Keep-Alive
```

- Ordem dos cabeçalhos;
- No cabeçalho *Accept-Language*, apenas "pt";
- Inclusão do cabeçalho *UA-CPU*.

- *Safari 5*:

```
GET image.jpg HTTP/1.1
Accept-Encoding: gzip, deflate
Accept-Language: pt-PT
Referer: http://www.referer.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; pt-PT)
AppleWebKit/?523.15 (KHTML, like Gecko) Version/3.0 Safari/523.15
Accept: */*
Connection: keep-alive
Host: www.host.com
```

- Ordem dos cabeçalhos;
- No cabeçalho *Connection*, "*keep-alive*" com 'k' e 'a' minúsculos.

As características destacadas na comparação dos pedidos são as escolhidas para a criação dos perfis, listando-se a seguir:

- Ordem dos cabeçalhos;
- *Accept;*
- *Accept-Language;*
- *Accept-Encoding;*

- *Accept-Charset;*
- *Keep-Alive;*
- *Connection;*
- *TE;*
- *UA-CPU.*

Numa primeira experiência, o BrowserProfiles é executado com os parâmetros **–b** e **–t** a zero. A tabela 3.2.1.1. mostra os perfis criados e o respectivo número de ocorrências.

Tabela 3.2.2.1. - Perfis criados aplicando o valor 0 aos parâmetros **–b** e **–t**. A coluna *Count* contém o número de ocorrências de cada perfil.

Browser Name	Browser Version	Layout Engine	Count
Firefox	1	Gecko	1236
Firefox	2	Gecko	66445
Firefox	3	Gecko	598
Iceweasel	2	Gecko	86
Konqueror	3	KHTML	69
Minefield	3	Gecko	7
MSIE	2	Trident	15
MSIE	5	Trident	732
MSIE	6	Trident	152159
MSIE	7	Trident	142304
MSIE	8	Trident	13
Opera	9	Presto	2891
Other	Other	/	7312
Safari	5	WebKit	923
SeaMonkey	1	Gecko	69
Thunderbird	2	Gecko	37

Observando a tabela 3.2.1.1., pode-se concluir que, para serem mantidas as versões mais populares dos *browsers* mais populares e serem eliminados perfis com baixa base de conhecimento, o valor mais adequado para o parâmetro **-b** deve ser superior a 86 (*Iceweasel 2*) e inferior a 598 (*Firefox 3*). O valor escolhido foi 100. O perfil *Other* corresponde aos browsers, cujo a UAS não foi identificada pelo AuditRec. *Other* é inútil como perfil, porque contém características de todos os *browsers* que não foram identificados, por isso também não deve ser considerado. As versões *1* e *2* do *Firefox* não apresentam qualquer diferença nas características HTTP observadas, daí não se considerar a versão *1*, que é das duas a menos popular. Para **–t**, o valor escolhido também foi 100. Assim sendo, os perfis ficam reduzidos aos da tabela 3.2.1.2.

Tabela 3.2.2.2. - Perfis filtrados por atribuição do valor 100 aos parâmetros **–b** e **–t** e não consideração dos perfis *Other* e *Firefox 1*.

Browser Name	Browser Version	Layout Engine	Count
Firefox	2	Gecko	66445
Firefox	3	Gecko	598
MSIE	5	Trident	732
MSIE	6	Trident	152159
MSIE	7	Trident	142304
Opera	9	Presto	2891
Safari	5	WebKit	923

Os perfis não considerados *Iceweasel 2, Konqueror 3, Minefield 3, MSIE 2, MSIE 8, Other, SeaMonkey 1* e *Thunderbird 2* correspondem a 8060 ocorrências, o que representa apenas 2,03 % do total de ocorrências. Da não consideração desses perfis, pode-se concluir:

- Os perfis *Iceweasel 2, Minefield 3, SeaMonkey 1* e *Thunderbird 2* incorporam a *Layout Engine Gecko*. Sendo que é a *Layout Engine* que determina o comportamento dos *browsers*, no que diz respeito aos aspectos tratados no reconhecimento de browsers, não é possível distinguir *browsers* que utilizem a mesma *Layout Engine*. Todos os cliques provenientes de *browsers* que incorporam *Gecko* vão ser reconhecidos como *Firefox*.

- Os cliques provenientes do *browser Firefox 1* vão ser reconhecidos como provenientes de *Firefox 2.*

- A versão do *MSIE standard* actualmente é a *7*. A versão *6* ainda é muito utilizada e a *5* tem uma fatia de utilizadores residual. Versões anteriores à 5 não têm qualquer expressão actualmente. A versão 8 já existe, mas apenas em versão beta, o que explica a sua quase inexistência no tráfego analisado.

- A eliminação do perfil *Konqueror 3* é o único real prejuízo, porque a *Layout Engine KHTML* fica de fora dos perfis.

A colocação do valor 100 no parâmetro –t a 100 reduz o número de variantes de 1.094 para 319.

Em conclusão, pode-se dizer que incluídas nos perfis as actuais versões das *Layout Engines Gecko, Presto, Trident e WebKit*, a grande maioria do tráfego é reconhecível, sendo que só fica de fora uma *layout engine* com muito baixa representatividade.

3.2.3. Subconjunto DOM

DOM é um *standard* recomendado pelo W3C. Os fabricantes de *browsers* não seguem os *standards* recomendados plenamente, nuns casos por opção, noutros casos por desactualização. Isso significa que cada *browser* tem uma compatibilidade diferente com as funcionalidades recomendadas pelos *standards*. Essa diferença pode ser explorada no que se designa por detecção de objectos para identificação de *browsers*, que se processa da seguinte forma:

1. Estudo de compatibilidade dos *browsers* com as funcionalidades (propriedades e métodos) propostas pelo *standard*;
2. Selecção de funcionalidades não suportadas por todos os *browsers*;
3. Constituição de perfis de compatibilidade dos *browsers* com as funcionalidades seleccionadas;
4. Desenvolvimento de um *script* que invoca as funcionalidades seleccionadas ao compilador do *browser*. O compilador retorna '1', caso seja compatível com a funcionalidade invocada e '0', caso contrário.
5. Os resultados das invocações são comparados com os perfis. Caso ocorra uma correspondência, é retornado o nome do *browser* correspondente ao perfil.

O procedimento exemplifica-se através de um *standard* exemplo com cinco funcionalidades, nomeadamente F1, F2, F3, F4 e F5 e três *browsers* exemplo, nomeadamente B1, B2 e B3 e decorre da seguinte forma:

1. A tabela 3.2.2.1 fornece a compatibilidade dos *browsers* com as funcionalidades do *standard* exemplo.
2. Selecção das funcionalidades não suportadas por todos os *browsers*: F2, F3 e F4.
3. A tabela 3.2.2.2. contém os perfis dos *browsers* (PBs) criados;
4. *Script*:

```
var compatibilidade = "";

if( typeof(F2) != 'undefined' ) { compatibilidade += "1"; }
else { compatibilidade += "0"; }

if( typeof(F3) != 'undefined' ) { compatibilidade += "1"; }
else { compatibilidade += "0"; }

if( typeof(F4) != 'undefined' ) { compatibilidade += "1";}
else { compatibilidade += "0"; }
```

5. Comparar cada PB com "compatibilidade" até ocorrer uma correspondência. Exemplo:

```
compatibilidade = "100" => PB1
```

Tabela 3.2.3.1. - Compatibilidade dos *browsers* exemplo B1, B2 e B3 com as funcionalidades F1, F2, F3, F4 e F5 do *standard* exemplo. (1 - Compatível, 0 – Não compatível)

Funcionalidade	**B1**	**B2**	**B3**
F1	1	1	1
F2	1	0	1
F3	0	1	1
F4	0	0	1
F5	0	0	0

Tabela 3.2.3.2. - Perfis de compatibilidade dos *browsers* com as funcionalidades seleccionadas. (PBi – Perfil do *Browser* i, 1 - Compatível, 0 – Não compatível)

Funcionalidade	**PB1**	**PB2**	**PB3**
F2	1	0	1
F3	0	1	1
F4	0	0	1

A tabela 3.2.2.3. contém as funcionalidades DOM usadas na detecção de objectos que é feita na interacção *JavaScript* descrita na subsecção 3.2.4.

Tabela 3.2.3.3. - Funcionalidades (propriedades e métodos) usadas na detecção de objectos para reconhecimento de *browsers*. Adaptada de [4] (1 - Compatível, 0 – Não compatível)

Funcionalidade	***Firefox 2***	***Firefox 3***	***MSIE 5***	***MSIE 6***	***MSIE 7***	***Opera 9***	***Safari 5***
document.all	0	0	1	1	1	0	0
document.childNodes	1	1	0	0	0	1	1
document.compatMode	1	1	0	1	1	1	1
document.documentMode	0	0	1	1	0	0	0
navigator.accentColorName	0	0	1	1	1	0	0
navigator.product == 'Gecko'	1	1	0	0	0	0	0
navigator.savePreferences	0	0	1	1	1	0	0
navigator.taintEnabled()	1	1	0	0	0	0	0
window.find	1	1	0	0	0	1	1
window.opera	0	0	0	0	0	1	0
window.XMLHttpRequest	1	1	0	0	1	1	1
document.activeElement	0	1	1	1	1	1	0
document.charset	0	0	1	1	1	1	1
document.elementFromPoint()	0	1	1	1	1	1	1
document.enableStyleSheetsForSet()	0	1	0	0	0	0	0
window.content	1	1	0	0	0	0	0
window.atob()	1	1	0	0	0	0	1
navigator.appMinorVersion	0	0	1	1	1	1	0
screen.availLeft	1	1	0	0	0	0	1
screen.bufferDepth	0	0	1	1	1	0	0
screen.left	1	1	0	0	0	0	0

3.3. Programa de Reconhecimento

Acrescentando à figura 3.2.1. a componente de reconhecimento resulta a figura 3.3.1. AuditProfiler, o programa de reconhecimento compara o *browser* a reconhecer (*Browser to be recognized*) com cada um dos perfis presentes na base de dados (*Database*). De cada comparação resulta uma pontuação (*Score*), que é tanto mais alta quanto maior for a semelhança entre o *browser* e o perfil. No exemplo da figura, a escala de pontuação está compreendida entre zero e dois. As duas características têm o mesmo peso; vale um ponto cada. O perfil com mais alta pontuação corresponde ao resultado do reconhecimento. Na tabela da figura (*Profiling Results*), com a pontuação atribuída a cada perfil ordenada decrescentemente, pode-se observar como o *browser* com forma circular e cor vermelha obteve uma correspondência máxima com o perfil *Browser 2*.

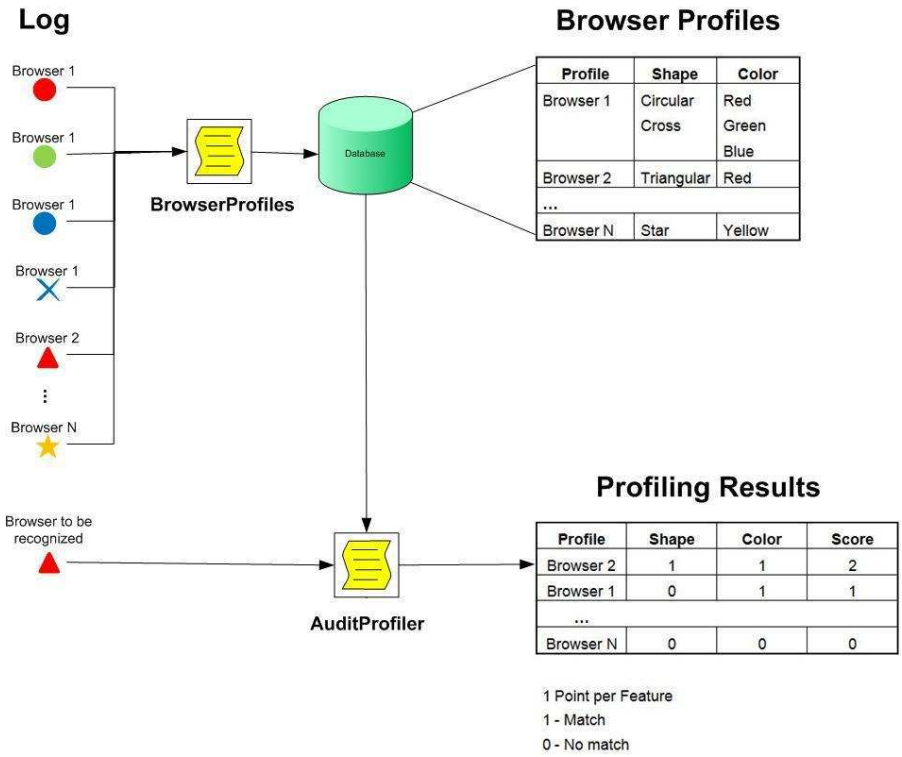

Figura 3.3.1. - Diagrama do sistema de sistema de reconhecimento, incluindo as componentes de criação de perfis e reconhecimento.

O AuditProfiler e processa os seguintes campos do *log* (formato AuditRec versão *online*):

- HTTP:
    - Header Lines Order;
    - Accept;
    - Accept-Language;

    - Accept-Encoding;
    - Accept-Charset;
    - Keep-Alive;
    - Connection;
    - TE;
    - UA-CPU.
- Object Detection (DOM);
- Integridade e segurança:
    - TCP Src (tres.php);
    - Magic Cookie;
    - Magic Cookie Validation;
    - CRC (dois.php);
    - CRC (tres.php).
- Contrato:
    - Referer do anunciante;
    - Referer do expositor.

Os campos HTTP e *object detection* são processados pelo algoritmo de reconhecimento de *browsers*. A cada campo foi atribuída uma pontuação, que pode ser consultada na tabela 3.3.1.

Tabela 3.3.2. – Pontuação por característica. Um ponto por subconjunto, perfazendo um total de dois pontos.

Subconjunto	Característica	Pontuação
HTTP	Header Lines	0.2
	Accept	0.1
	Accept-Language	0.1
	Accept-Encoding	0.1
	Accept-Charset	0.1
	Keep-Alive	0.1
	Connection	0.1
	TE	0.1
	UA-CPU	0.1
DOM	Object Detection	1

Descrição do algoritmo de reconhecimento, tal como ilustrado na figura 3.3.1.:

Para cada clique (*Click*):

- Inserir conteúdo HTTP e DOM do clique num vector, um campo em cada posição de vector.
- Para cada perfil (*Profile*):
    - Para cada caracterísitica (*Feature*):
        - Passar o conteúdo do campo da base de dados para um vector, uma variante em cada posição de vector.
        - Comparar cada variante com o conteúdo da posição do vector correspondente ao campo seleccionado.
            - Se ocorrer uma igualdade (*Match*) é somada à pontuação do perfil (*Score*), a pontuação referente à característica correspondida. Passar à próxima característica.

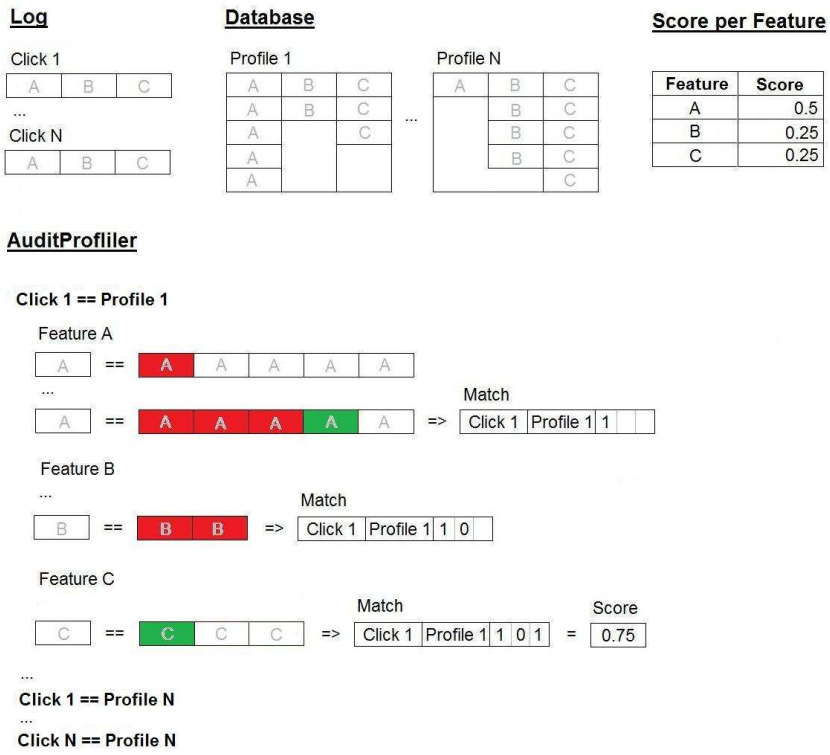

Figura 3.3.1. – Esquema de funcionamento do algoritmo de reconhecimento do programa AuditProfiler.

3.4. Serviço de Auditoria

Tal como se pode observar pela figura 3.4.1., o serviço de auditoria é constituído por quatro elementos: 1) servidor auditor; 2) servidor do anunciante; 3) servidor do expositor; 4) cliente. Cada clique do cliente no anúncio colocado no expositor é auditado estabelecendo uma interacção *JavaScript* entre o cliente e o servidor auditor. Os pedidos HTTP envolvidos na interacção *JavaScript* são capturados em tempo real pelo AuditRec versão *online* e processados de hora a hora pelo AuditProfiler.

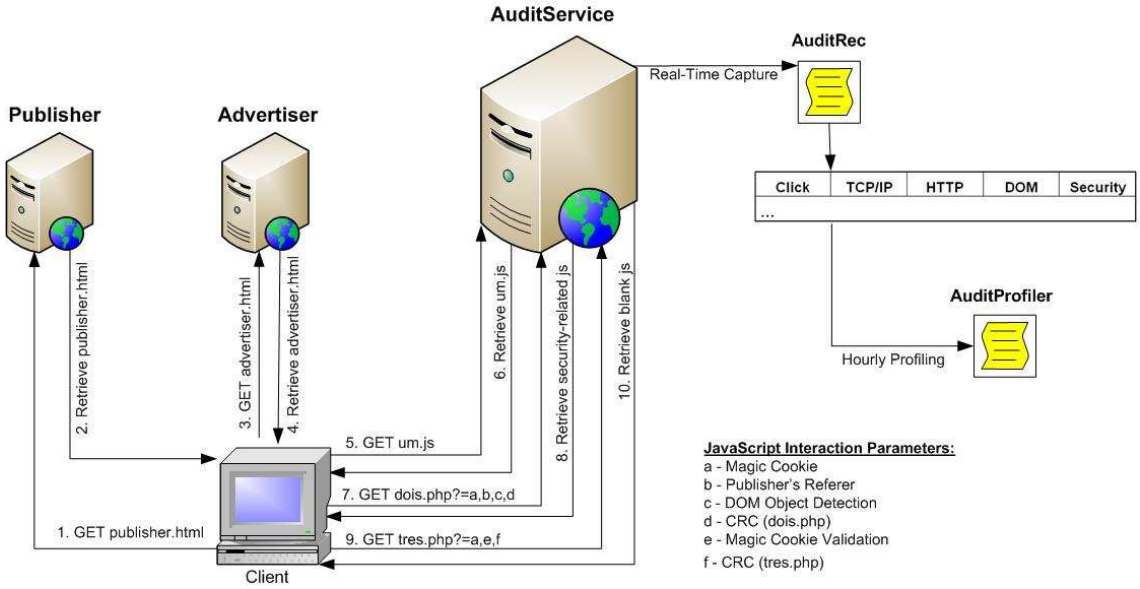

Figura 3.4.1. – Diagrama da interacção *JavaScript* entre cliente e serviço desencadeada por um clique realizado num anúncio. Captura dos cliques em tempo real e processamento de reconhecimento e validação de hora a hora.

A figura ilustra a interacção *JavaScript* e como é desencadeada. O cliente (*Client*) começa por introduzir o endereço do expositor (*Publisher*) na barra de endereços do *browser*. Endereço do expositor:

```
http://www.publisher.com/
```

O *browser* faz um pedido HTTP do documento HTML ao servidor do expositor, o que lhe é retornado. O *browser* carrega o conteúdo do documento, que contém um anúncio com uma hiperligação de publicidade para o sítio do anunciante (*Advertiser*). O código HTML respectivo é o seguinte:

```
<a href="http://www.advertiser.com/">
<img src="img/advertiser.jpg"></a>
```

Quando um cliente dá um clique no anúncio, o *browser* faz um pedido HTTP da página do anunciante, ao servidor do mesmo. Nesse pedido é incluído o cabeçalho *Referer* com o endereço do expositor. O documento de resposta devolvido pelo servidor do anunciante contém embebido um *javascript* denominado *um.js*, que se encontra alojado no servidor do serviço de auditoria (*AuditService*). O código HTML respectivo é o seguinte:

```
<script type="text/javascript" src="http://www.auditservice.com/um.js">
</script>
```

O cliente faz o pedido HTTP do *um.js* ao *AuditService*. O *AuditService* devolve o mesmo, que contém o teste de detecção de objectos DOM e uma instrução para pedir o *dois.php,* também ao *AuditService*. O cliente faz o pedido HTTP do *dois.php*, passando os seguintes parâmetros no GET do pedido:

- a – *Magic Cookie*,
- b – *Referer* da página que estabeleceu a hiperligação com a página do anunciante, em que o um.js está embebido, neste caso: http://www.publisher.com/;
- c – Resultado do teste de detecção de objectos DOM;
- d – CRC.

O *dois.php* gera uma validação da *Magic Cookie* contida no parâmetro 'a'. O par Magic Cookie validação gerada é guardado na base de dados para posterior verificação. O dois.php devolve um *JavaScript* específico para o browser do cliente. Esse *JavaScript* contém a validação da Magic Cookie e uma linha que efectua o pedido do *tres.php*. O *tres.php* é pedido pelo cliente com os seguintes parâmetros no GET:

- a – *Magic Cookie*;
- e – *Magic Cookie Validation*;
- f – CRC.

Por razões de segurança, todos os parâmetros excepto o 'c' são codificados em MD5 antes de serem transmitidos. Cada clique no anúncio gera três pedidos HTTP ao *AuditService*, mas apenas os pedidos do dois.php e do tres.php são considerados para o reconhecimento. Isto, porque o primeiro pedido não inclui parâmetros, o que significa que pode não ser efectuado, caso o utilizador tenha visitado o sítio do anunciante recentemente. O *browser* não repete o pedido, porque já tem a sua resposta em *cache*. Para se assegurar que um pedido é efectuado, tem de se incluir um parâmetro designado por *Magic Cookie*, cujo valor seja diferente em cada pedido. Para se garantir que a *Magic Cookie* é diferente em cada pedido, é gerada a partir do tempo (data e hora à resolução de ms). A *Magic Cookie* adquire a segunda função de servir de identificação para o clique. O pedido do tres.php existe apenas por razões de segurança, especialmente contra ferramentas de clique automático que realizam pedidos independentemente das respostas, porque não são capazes de as interpretar. Para um clique ser validado, o parâmetro 'e' do pedido do tres.php tem de conter a validação da *Magic Cookie* enviada ao cliente como resposta ao pedido do dois.php. Os pedidos dois.php e tres.php têm um parâmetro de controlo de integridade (CRC), cuja finalidade é verificar se os parâmetros foram alterados durante a transmissão. O CRC consiste em gerar uma *hash* MD5 dos parâmetros. A verificação é posteriormente feita pelo programa de reconhecimento, que também gera uma *hash* MD5 dos parâmetros e compara-a com a *hash* recebida no campo CRC.

O AuditProfiler é adaptado ao serviço de auditoria passando a processar também os parâmetros da seguinte forma:

- Lê o campo *TCP Src (tres.php)*. Se o seu conteúdo for nulo significa que o pedido do tres.php não foi efectuado. Sendo que a validação de segurança é transmitida no pedido do tres.php, o clique não passa o teste de segurança.

- Lê os campos *Magic Cookie* e *Validation*. Verifica se esse par *Magic Cookie Validation* existe na base de dados. Caso não exista, o clique não passa o teste de segurança.

- Associa um contador a todas as *Magic Cookies* presentes no *log*. Todos os cliques, cujas *Magic Cookies* tenham o respectivo contador com um valor superior a 1, não passam o teste de segurança.

- O teste de integridade (CRC) é feito ao conteúdo dos parâmetros transmitidos nos pedidos do dois.php e do tres.php para verificar se não foram alterados durante a transmissão. O último parâmetro de cada pedido contém uma *hash* MD5 gerada a partir do conteúdo dos restantes parâmetros. O AuditProfiler gera também uma *hash* desses parâmetros e compara-a com a que foi transmitida. Se a comparação não o clique não passa a verificação de integridade.

- Identifica a que contrato de publicidade está associado o clique. Um contrato é constituído por um par anunciante expositor. O AuditProfiler lê do cabeçalho *Referer* do pedido HTTP o nome do anunciante, e do parâmetro *Publisher* do GET o nome do expositor. Depois procura na base de dados o identificador do contrato correspondente ao par anunciante expositor lido.

3.5. Interface *Web* de Resultados

A interface *Web* desenvolvida em linguagem PHP chama-se AuditInterface. Tem como função tornar acessíveis e inteligíveis de forma estatística os resultados de reconhecimento produzidos pelo AuditProfiler. Na figura 3.5.1 pode-se ver a página inicial da AuditInterface, onde é seleccionado o conjunto de cliques a analisar. A selecção pode ser feita por captura, por contrato ou por captura e contrato.

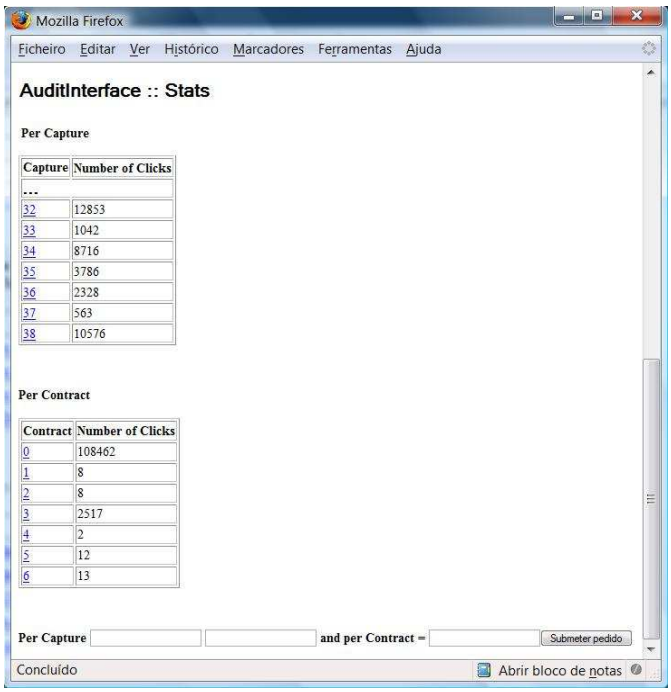

Figura 3.5.1. - Página inicial da AuditInterface, onde é seleccionado o conjunto de cliques a analisar.

A figura 3.5.2. exibe, como exemplo, a estatística para o conjunto de cliques pertencente à captura número 38, onde se podem ver três tabelas e uma informação geral. A informação geral dá o número total de cliques em análise e a média de pontuação dos cliques. A primeira tabela contém a contagem e percentagem de cliques que foram reconhecidos como cada um dos perfis existentes.

As tabelas 1) e 2) incluem testes de validação dos cliques. A tabela 1) apresenta a contagem e percentagem de cliques que se inserem nas seguintes condições:

- Pontuação inferior a três valores de referência: 2 (máxima possível), 1,7 e 0,7.
- Não conformidade entre o *browser* (nome e versão), tal como se identifica pelo campo *User-Agent* (*Browser(UA)*) que envia no pedido HTTP, e o perfil de maior correspondência (*Browser(MaxScore)*).

A tabela 2) contém a contagem e percentagem de cliques para as seguintes verificações de integridade e segurança:

- Se o tres.php foi pedido;
- Se o par *Magic Cookie* código de validação constituído pelos parâmetros ‘a’ e ‘e’ do pedido do tres.php, corresponde ao par que foi inserido na base de dados;
- Se alguma *Magic Cookie* foi repetida;
- Se a verificação CRC do pedido do dois.php falhou;
- Se a verificação CRC do pedido do tres.php falhou.

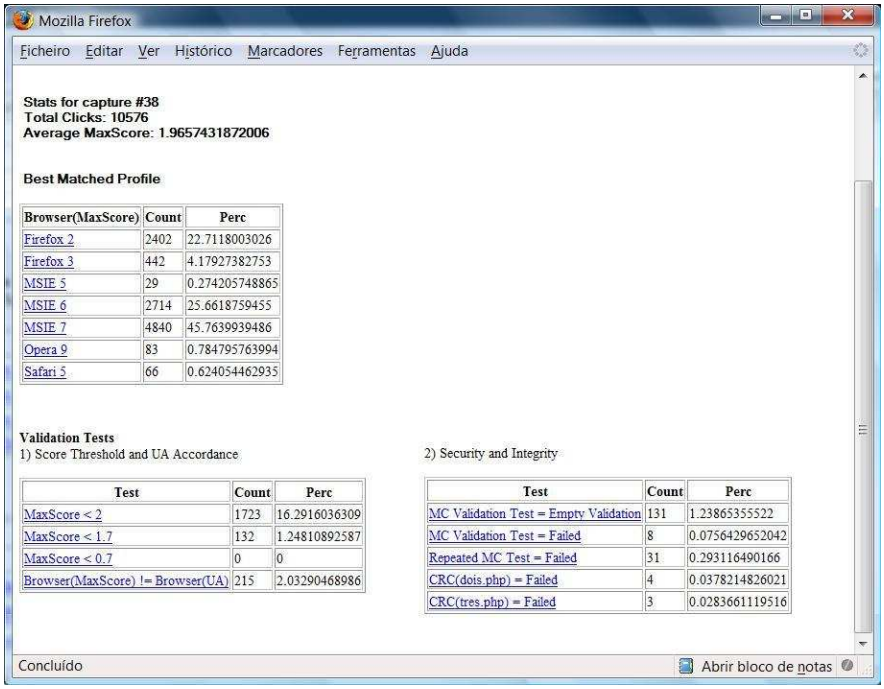

Figura 3.5.2. - Exemplo de uma página de estatísticas da AuditInterface.

Cada estatística tem uma hiperligação para os detalhes de captura e reconhecimento dos cliques abrangidos. A figura 3.5.3 dá como exemplo a consulta da estatística *Browser(MaxScore) = Firefox 2.* Podem-se ver listados os cliques que foram reconhecidos como produzidos pelo *browser Firefox 2* e os respectivos resultados de verificações, pontuação e correspondência com a informação presente na UAS. Cada clique tem associadas duas hiperligações: 1) para os seus detalhes de captura, 2) para a sua informação completa de reconhecimento, onde estão registadas as pontuações que o clique teve para os restantes perfis.

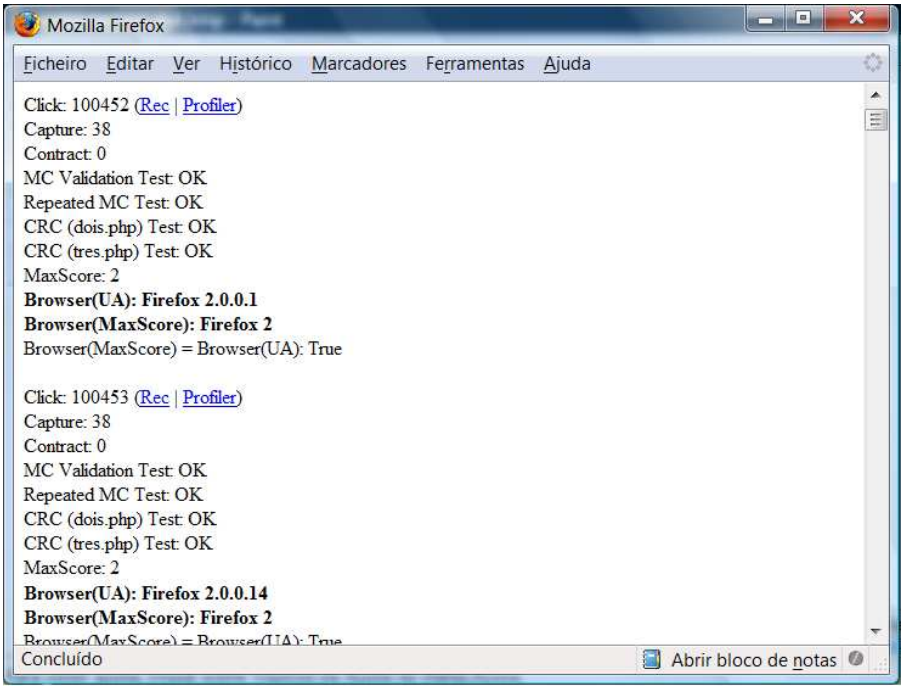

Figura 3.5.3. - Exemplo de uma página com detalhes de reconhecimento dos cliques abrangidos por uma determinada estatística.

A figura 3.5.4. contém a página de detalhes de captura acedida pela hiperligação *Rec*. A página não contém todos os detalhes de captura presentes no *log*, apenas os mais relevantes. Por razões de privacidade, as informações de endereço de IP de origem, *Host* e *Referer* foram ofuscadas. A cinzento (não pertencente à página) são indicadas as proveniências das informações: 1) cabeçalho TCP/IP; 2) cabeçalho HTTP; 3) parâmetros transmitidos na interacção *JavaScript*.

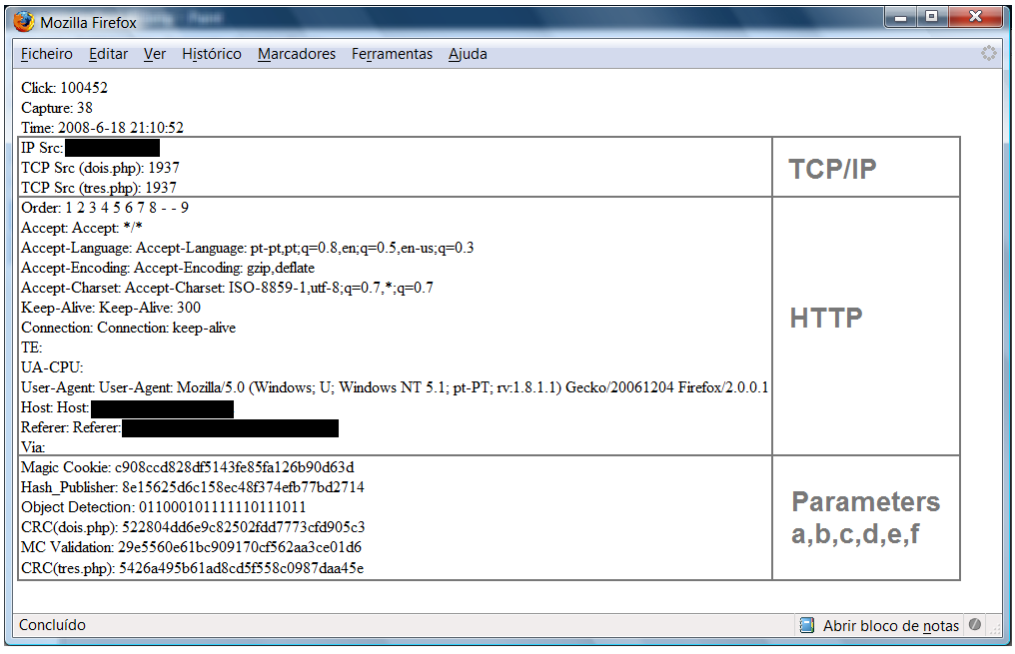

Figura 3.5.4. - Exemplo de uma página com detalhes de captura de um clique.

A figura 3.5.5. contém a página de informação de reconhecimento completa acedida pela hiperligação *Profiler*. Podem-se observar as pontuações obtidas pelos restantes perfis ordenadas decrescentemente e que características contribuíram para a pontuação.

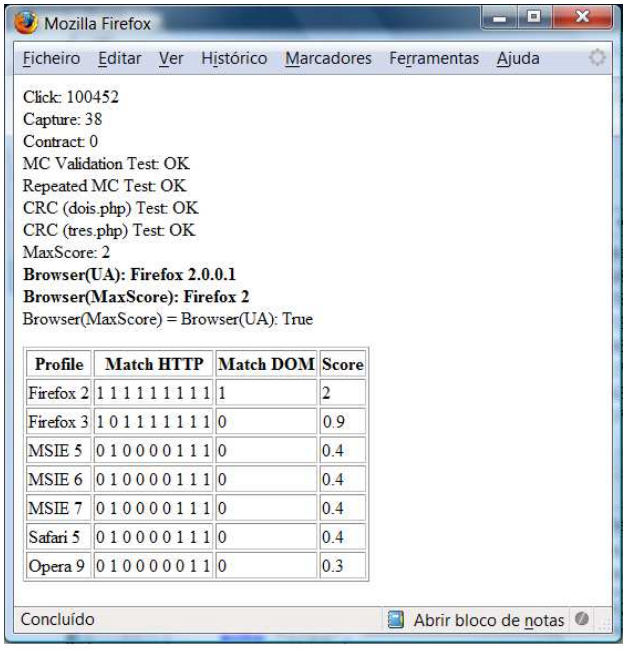

Figura 3.5.5. - Exemplo de uma página com informação de reconhecimento completa de um clique.

3.6. Conclusão

Comparando a solução implementada com o projecto *browserrecon* apresentado no estado da arte, retiram-se as seguintes vantagens:

- Acesso ao pedido HTTP na sua forma original, tal como captada na rede. No *browserrecon* os cabeçalhos HTTP dos pedidos são acedidos pela função PHP *getallheaders()*, que não regista a ordem em que os cabeçalhos foram enviados. A ordem é uma informação muito rica, que não deve ser negligenciada.
- Testes de compatibilidade DOM;
- Filtragem dos perfis:
    - Os perfis foram criados tendo como critério que todas as *layout engines* devessem ser incluídas e representadas pelo *browser* mais popular que as incorpora. Assim evitam-se situações de ambiguidade no reconhecimento, tal como se verifica no *browserrecon*.
    - Alguns *browsers* têm actualizações periódicas, que resultam em novas sub-versões, por exemplo *Firefox 2.0.0.1*. Estas actualizações não trazem evoluções em termos de HTTP ou DOM, apenas resolvem algumas

vulnerabilidades de segurança. No nosso projecto os perfis foram feitos pela versão *(Firefox 2*, por exemplo) e não pela sub-versão, como no *browserrecon*. Mais uma vez, se evitam ambiguidades e se diminui o tempo de processamento, já que é reduzido o número de comparações (desnecessárias, neste caso) a realizar.

- Facilidade de instalação. A instalação do nosso sistema de reconhecimento numa página requer apenas a adição de uma linha de código HTML à mesma. Essa linha é necessária para estabelecer a interacção *JavaScript* com o nosso servidor. Recorda-se a linha:

```
<script type="text/javascript" src="http://www.auditservice.com/um.js"></script>
```

4. Testes

4.1. Introdução

Neste capítulo são descritos os testes realizados ao sistema de reconhecimento e analisados os resultados produzidos. Os testes são os seguintes:

1. Teste cliques manuais em sítio *Web* experimental;
2. Teste cliques provenientes de um sítio *Web* real;
3. Teste cliques provenientes de ferramentas de clique automático.

O primeiro teste consiste no reconhecimento de cliques realizados pelos *browsers* mais populares. Também são feitos cliques com a UAS alterada. O objectivo do teste é verificar se os *browsers* são correctamente reconhecidos e se o reconhecimento é efectivamente independente da informação contida na UAS. No segundo teste, com tráfego real, avalia-se a resistência e abrangência do sistema a um grande volume de tráfego. No terceiro teste avalia-se a eficácia do método na detecção de cliques fraudulentos.

4.2. Cenário de Teste

O cenário de teste simula um serviço de auditoria constituído por quatro elementos: o servidor do auditor, o servidor do anunciante, o servidor do expositor e o cliente. O anunciante tem um contrato de publicidade com o expositor. O anúncio inserido no sítio do expositor é uma imagem com uma hiperligação para o sítio do anunciante. O anunciante encomenda um serviço de auditoria para avaliar a qualidade do tráfego proveniente do expositor. O cliente acede ao sítio do expositor, clica no anúncio e é reencaminhado para o anunciante. Automaticamente, é iniciada uma interacção com o servidor do auditor.

O servidor do auditor chama-se *AuditSrv* e é composto por duas partes: 1) sistema de reconhecimento de *browsers;* 2) servidor *Web*. O sistema de reconhecimento é constituído pelos programas de captura e reconhecimento e pela base de dados. Na base de dados estão registados os perfis, são guardados os cliques e os respectivos resultados de reconhecimento. O servidor *Web* aloja duas componentes, o *AuditService* e o *AuditInterface*. A componente

AuditService trata da interacção *JavaScript*. A componente *AuditInterface* os resultados de reconhecimento de *browsers* processados estatisticamente.

4.3. Teste Cliques Manuais em Sítio *Web* Experimental

O procedimento de teste é o seguinte:

1. Executar o ficheiro de captura para registar num *log* os pedidos feitos pelo cliente ao AuditService;
2. Para cada browser e para cada UAS, aceder ao sítio do anunciante (*Publisher*) e dar um clique na publicidade do anunciante;
3. Executar o ficheiro de reconhecimento de *browsers*;
4. Analisar os resultados do reconhecimento.

Conforme o estudo de mercado analisado no Capítulo 2 e conforme os perfis criados, escolheram-se os seguintes *browsers* para o teste: *Firefox 2, Internet Explorer 7, Opera 9* e *Safari 5*. Instalaram-se *add-ons* de alteração da UAS nos *browsers Firefox 2* e *Safari 5*. Não existem *add-ons* desse género para os *browsers MSIE 7 e Opera 9*. Fizeram-se seis cliques com o *Firefox 2*, um com o *MSIE 7*, um com o *Opera 9* e cinco com o *Safari 5*. A tabela 4.3.1. contém a sequência de cliques efectuados.

Tabela 4.3.1. - Browsers e respectivos UASs utilizados no teste de cliques manuais.

Click	Browser	UAS
1	Firefox 2	Default
2	Firefox 2	Firefox 3
3	Firefox 2	MSIE 6
4	Firefox 2	MSIE 7
5	Firefox 2	Opera 9
6	Firefox 2	Safari 5
7	MSIE 7	Default
8	Opera 9	Default
9	Safari 5	Default
10	Safari 5	Firefox 2
11	Safari 5	MSIE 6
12	Safari 5	MSIE 7
13	Safari 5	Opera 9

Cada clique, com todas as características HTTP e DOM recolhidas, é comparado com cada um dos sete perfis. Dessa comparação resulta uma pontuação, que é tanto mais alta quanto maior for a semelhança entre o clique e o perfil. O perfil com maior pontuação, *Browser(MaxScore)*, indica, segundo o reconhecimento realizado, qual o browser que com maior probabilidade foi utilizado para produzir o clique.

Tabela 4.3.2. – Estatísticas do reconhecimento de cliques realizados manualmente.

Total Clicks: 13
Average MaxScore: 2

Test	Count	Perc
Browser(MaxScore) = Firefox 2	6	46.1538461538
Browser(MaxScore) = Firefox 3	0	0
Browser(MaxScore) = MSIE 5	0	0
Browser(MaxScore) = MSIE 6	0	0
Browser(MaxScore) = MSIE 7	1	7.69230769231
Browser(MaxScore) = Opera 9	1	7.69230769231
Browser(MaxScore) = Safari 5	5	38.4615384615
Browser(MaxScore) != Browser(UA)	9	69.2307692308

Os dados apresentados pela tabela 4.3.2. provam que o AuditProfiler reconheceu correctamente os cliques realizados: seis cliques de *Firefox 2*, um clique de *MSIE 7*, um clique de *Opera 9* e 5 cliques de *Safari 5*. A média da pontuação dos 13 cliques foi de dois pontos, o que significa que todos obtiveram a pontuação máxima possível. Isso é relevante, porque torna inequívoca a validade dos cliques. A correspondência entre clique e perfil é total, para as características em análise, como se pode observar pelo exemplo da tabela 4.3.3. Na tabela 4.3.3. a pontuação (*Score*) da correspondência das nove características HTTP e das duas componentes DOM, é de dois pontos. No exemplo, o perfil com maior pontuação é o *Firefox 2*. Pode-se ver quão grande é a diferenciação em relação a perfis referentes a outros *browsers* - 1,6 ou 1,7 pontos. Também se pode observar que entre versões do mesmo *browser* a diferenciação é muito menor, neste caso de apenas 0,4 pontos.

Tabela 4.3.3. – Pontuação para cada perfil de um clique realizado por Firefox 2.

Profile	Match HTTP	Match DOM	Score
Firefox 2	1 1 1 1 1 1 1 1 1	1	2
Firefox 3	1 0 1 1 1 1 1 1 1	0	0.9
MSIE 5	0 1 0 0 0 0 1 1 1	0	0.4
MSIE 6	0 1 0 0 0 0 1 1 1	0	0.4
MSIE 7	0 1 0 0 0 0 1 1 1	0	0.4
Safari 5	0 1 0 0 0 0 1 1 1	0	0.4
Opera 9	0 1 0 0 0 0 0 1 1	0	0.3

Quanto à comparação entre o resultado do reconhecimento e o browser anunciado pela UAS, não se verifica em nove casos, tal como esperado. Recordando, foram realizados nove cliques com a UAS alterada, 5 pelo *Firefox 2* e 4 pelo *Safari 5.*

A tabela 4.3.3. contém informação sobre a integridade dos cliques. Essa informação está relacionada com a segurança do serviço e não influencia a pontuação dos cliques.

Tabela 4.3.4.– Estatísticas de verificação de integridade, da informação devolvida pelo cliente, no teste de cliques manuais.

Test	Count	Perc
tres.php Not Requested	0	0
MC Validation Test = Failed	0	0
Repeated MC Test = Failed	0	0
CRC(dois.php) = Failed	0	0
CRC(tres.php) = Failed	0	0

Pode-se constatar da observação da tabela 4.3.3., que nenhum clique apresentou qualquer falha a nível das verificações de integridade, o que está de acordo com o esperado, já que se tratam de cliques legítimos.

4.4. Teste Cliques Provenientes de um Sítio *Web* Real

Este teste avalia o desempenho do sistema de reconhecimento num cenário em que é aplicado a tráfego de um sítio existente. O sítio *Web*, que colaborou neste teste, tem uma média de 7000 visitas diárias.

O cenário de teste inclui o cliente (*Client*), o servidor *Web* do sítio real (*RealSite*) e o servidor *Web* do AuditService (*AuditService*). Neste caso, *Client* representa os visitantes do sítio existente. O *um.js* é embebido no documento HTML da página principal do sítio real. O teste desenrola-se de forma análoga ao teste anterior, excepto que em vez dos sítios de expositor (*Publisher*) e anunciante (*Advertiser*), existe apenas o sítio real. O objectivo não é o de analisar uma transacção de publicidade online, mas o de medir a qualidade do tráfego que visita o servidor do sitio web, no que toca a integridade dos cliques. A figura 4.4.1. ilustra a configuração deste teste e a comunicação feita entre o cliente e os servidores.

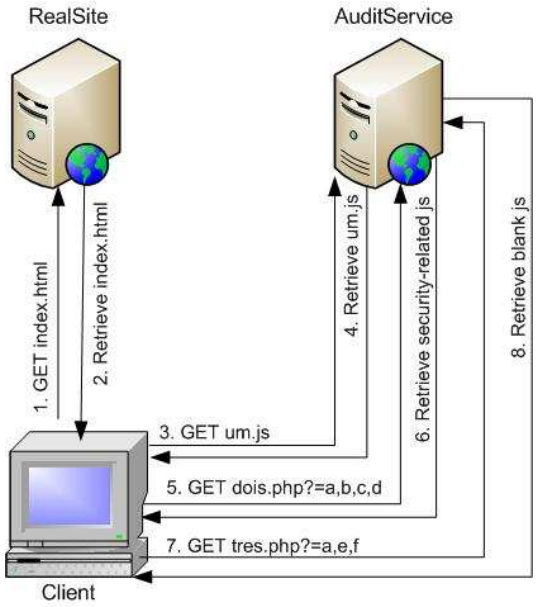

Figura 4.4.1. – Cenário do Teste Cliques de um Sítio *Web*.

Uma vez colocado o um.js no sítio *Web*, executa-se o ficheiro de captura. O ficheiro de reconhecimento é configurado para ser executado de hora a hora.

Os resultados do teste são acompanhados na *AuditInterface*, com actualizações de hora a hora. Ao fim de dez dias tinham sido capturados 71.058 cliques (pedidos HTTP). Na tabela 5.4.x retirada da *AuditInterface* estão as estatísticas finais do teste. Os resultados aproximam-se das quotas de mercado de *browsers* apresentadas no capítulo 2, e são os seguintes: *Firefox* com uma quota de 24,11%, *MSIE* com 74,02%, *Opera* com 1,18% e *Safari* com 0,68%. Uma das diferenças para o teste anterior, é que, para além do número de cliques analisados ser bastante superior, neste teste não existem dados sobre que *browsers* realizaram efectivamente os cliques em análise. Os resultados do reconhecimento apenas podem ser avaliados por comparação com a informação transmitida no campo UAS. A variável estatística *Browser(MaxScore) != Browser(UA),* presente na tabela 4.4.2*.,* representa a contagem e percentagem de cliques em que não houve igualdade entre o reconhecimento (*Browser(MaxScore)*) e o *browser* indicado na UAS (*Browser(UA)*). A desigualdade acontece em 1,13% dos cliques. A *AuditInterface* permite ainda ver os resultados detalhados de cada clique. Dos 802 cliques listados, 591 (0,83% do total) falharam apenas no reconhecimento da versão do *browser*, tendo acertado no nome do mesmo. Desses 591 cliques:

- 424 são *Firefox*, dos quais 274 são *Firefox 1* que foram reconhecidos como *Firefox 2*. *Firefox 1* e *Firefox 2* são indistinguíveis, nos aspectos HTTP e DOM tratados pelo sistema de reconhecimento. Por essa razão não foi criado o perfil *Firefox 1*. Assumiu-se que o *Firefox 1* fosse reconhecido como *Firefox 2*. Os restantes 150 são *Firefox 2* que foram reconhecidos com *Firefox 3* e vice-versa.
- 111 são *MSIE*, em que a maioria são *MSIE 7* que foram reconhecidos como *MSIE 6*.
- 48 são *Safari 4*, que foram reconhecidos como *Safari 5*.
- 8 são *Opera 8*, que foram reconhecidos como *Opera 9*.

Os restantes 211 (0,30% do total) falharam, tanto no reconhecimento da versão como do nome do browser. Desses 211 cliques:

- A maioria utilizam o *Gecko*, o mesmo *Layout Engine* utilizado pelo *Firefox*: *Netscape*, *SeaMonkey*, *Iceweasel*, *Camino*, *Minefield*, *Galeon*, *Epiphany*, *Bon Echo*. Por esse motivo o seu comportamento assemelha-se ao *Firefox*, tendo sido reconhecidos pelo *AuditProfiler* como provenientes do *Firefox 2* ou *3*.

- 8 são *Konqueror*, cuja *Layout Engine KHTML*, não está presente nos perfis. Neste momento são reconhecidos como *Safari 5*, o que não é completamente errado, já que a *Apple*, fabricante da *layout engine WebKit* para *Safari*, faz parte da equipa produtora da *layout engine KHTML*.

- 16 têm a UAS em branco.

- Outros browsers, como por exemplo o da *Sony PlayStation*.

O segundo aspecto a analisar é a pontuação dos cliques. A média da pontuação dos 71.058 cliques analisados é de 1,97. Idealmente e se pudesse considerar que os cliques foram todos realizados por *browsers*, a média deveria ser de 2, que é o valor máximo possível. A pontuação máxima de 2 não foi atribuída a 15,21% dos cliques. Acrescentaram-se as seguintes duas colunas com estatísticas à tabela para ajudar a explicar este valor: número de cliques cuja ordem dos cabeçalhos HTTP não corresponde à ordem do perfil, e desses, quantos passaram por um servidor *proxy*. A ordem não teve correspondência em 10,04% dos cliques, sendo que 6,49% passaram por servidores *proxy*. Os servidores *proxy* acrescentam cabeçalhos aos pedidos HTTP que encaminham, alterando a sua ordem original. Um dos cabeçalhos acrescentados é o *Via*, onde é registado o nome do servidor *proxy*. Se um pedido contiver cabeçalho *Via* (*Via != Empty*), quer dizer que foi encaminhado por um servidor *proxy*. Os restantes 3,55% apresentam ordens que não estão presentes nos perfis. Isto, pelo facto de serem pouco frequentes e não terem sido abrangidas pelo tráfego reunido para a criação dos perfis. Os restantes 5,17% de cliques com pontuação inferior a dois apresentam, na sua maioria, diferenças nos campos HTTP de *Accept* ou *Accept-Language* ou em ambos. Mais uma vez, a razão é a insuficiência do tráfego que serviu de base para a criação dos perfis, que se manifesta nas características HTTP com maior variabilidade, como a ordem dos campos, o campo *Accept* e o campo *Accept-Language*. As restantes características HTTP e DOM têm na sua grande maioria correspondência nos perfis. Na tabela 4.4.2. pode-se ver como 99,67% dos cliques obtêm pontuações iguais ou superiores a 1,5 pontos, sendo que 97,46% obtêm 1,8 pontos ou mais. Isso significa que as falhas existentes nos perfis afectam a pontuação de 99,67% dos cliques até 0,5 pontos, o que apesar de poder representar um erro de 25%, não afecta significativamente o reconhecimento. Se se utilizar mais tráfego para criar os perfis, este problema é minimizado.

Tabela 4.4.1. - Estatísticas do reconhecimento de cliques no Teste Cliques de um Sítio *Web*.

Total Clicks: 71058
Average MaxScore: 1.9673632762458

Test	Count	Perc
Browser(MaxScore) = Firefox 2	16123	22.6899152805
Browser(MaxScore) = Firefox 3	1012	1.42418869093
Browser(MaxScore) = MSIE 5	121	0.170283430437
Browser(MaxScore) = MSIE 6	18867	26.5515494385
Browser(MaxScore) = MSIE 7	33607	47.2951673281
Browser(MaxScore) = Opera 9	842	1.18494750767
Browser(MaxScore) = Safari 5	486	0.683948323904
Browser(MaxScore) != Browser(UA)	802	1.12865546455
MaxScore < 2	10810	15.2129246531
MaxScore < 1.7	736	1.0357735934
MaxScore < 1.5	236	0.332123054406
Order(MaxScore) Match = 0	7138	10.0453150947
Order(MaxScore) Match = 0 AND Via != Empty	4611	6.48906527062

As estatísticas de verificação de integridade disponíveis na *AuditInterface*, podem ser analisadas na tabela 4.4.3. Na *AuditInterface* pode-se consultar a listagem dos cliques, com informação completa de reconhecimento, que falharam determinada verificação, clicando na hiperligação da respectiva estatística. A listagem é útil para se procurarem explicações para as verificações falhadas. Começando pela verificação de cliques que pedem o *dois.php*, mas não pedem o *tres.php*. Analisando a lista dos cliques que falharam a verificação, não há evidência de ser predominante em qualquer dos browser. A falha tem uma distribuição pelos browsers, semelhante à distribuição da captura completa. Uma possível explicação para estas falhas é: encerramento do browser ou *tab*, em que está a ser visualizada a página, que tem o *um.js* embebido, imediatamente após o pedido do *dois.php*. Analisando as estatísticas relacionadas com as *Magic Cookies* e começando pelas repetidas, verifica-se que nenhuma *Magic Cookie* é repetida mais do que uma vez. A *Magic Cookie* é gerada a partir da data e hora, à resolução de milissegundo, no momento do clique. Nas horas de maior tráfego é possível que ocorram cliques simultâneos (no mesmo milissegundo). Consequentemente, existirão na base de dados duas validações para a mesma Magic Cookie, o que explica que algumas verificações de validação tenham falhado. Quanto às verificações de não alteração do conteúdo dos parâmetros (CRC), o número de falhas encontradas é praticamente nulo, portanto desprezável.

Tabela 4.4.2. - Estatísticas de verificação de integridade, da informação devolvida pelo cliente, no Teste Cliques de um Sítio *Web*.

Test	Count	Perc
tres.php Not Requested	964	1.35663823918
MC Validation Test = Failed	67	0.0942891722255
Repeated MC Test = Failed	318	0.447521742802
CRC(dois.php) = Failed	13	0.0182949140139
CRC(tres.php) = Failed	11	0.0154803118579

4.5. Teste Cliques Provenientes de Ferramentas de Clique Automático

Este teste tem como objectivo verificar o desempenho do reconhecimento ao analisar tráfego proveniente de um sítio *Web* em conjunto com o tráfego proveniente de uma ferramenta de clique automático. Neste teste, os dois tráfegos foram gerados durante o mesmo período e ficaram registados na mesma captura.

O cenário deste teste contém os elementos do anterior, acrescido do *AuditClicker*, um computador com o *AuditClick* a ser executado. O *AuditClick* é uma ferramenta de clique automático, não disponível publicamente, e que pertence à empresa AuditMark.. O *AuditClick* inclui a maioria das funcionalidades geralmente disponíveis nas ferramentas de clique automático. Permite, entre outras coisas, simular o comportamento dos *browsers Firefox 2* e *Opera 9*, de forma aleatória.

O *AuditClick*, como a maioria das ferramentas de clique automático, não processa *JavaScript*. O *AuditClick* pede o *um.js*, mas não interpreta o seu conteúdo, não lendo o comando que pede o *dois.php*. Sendo assim, os cliques que produz no sítio *Web* não chegam a ser processados pelo sistema de reconhecimento, nunca sendo validados, o que provaria imediatamente a eficácia do sistema. Numa situação desse género, os dados recolhidos pelo sistema de reconhecimento seriam cruzados com o *log* de pedidos ao servidor do sítio *Web*, e os cliques que não resultassem em pedidos do dois.php, seriam considerados como inválidos. Pelo facto de não termos obtido autorização para produzir tráfego falso sobre um sítio *Web* que se encontra em produção, testámos o desempenho do sistema na situação mais desfavorável, e menos provável de acontecer, que é quando a ferramenta de clique automático não necessita de processar *JavaScript*. O *AuditClick* foi, então, programado para fazer os pedidos directamente ao *AuditService* e preencher os parâmetros dos pedidos com os valores correctos para o *browser* que pretende emular. O AuditClick ignora, portanto, o conteúdo ficheiros *JavaScript* que recebe como resposta aos seus pedidos, exceptuando uma informação: o parâmetro de validação da *magic cookie* tem de ser lido da resposta ao pedido do dois.php.

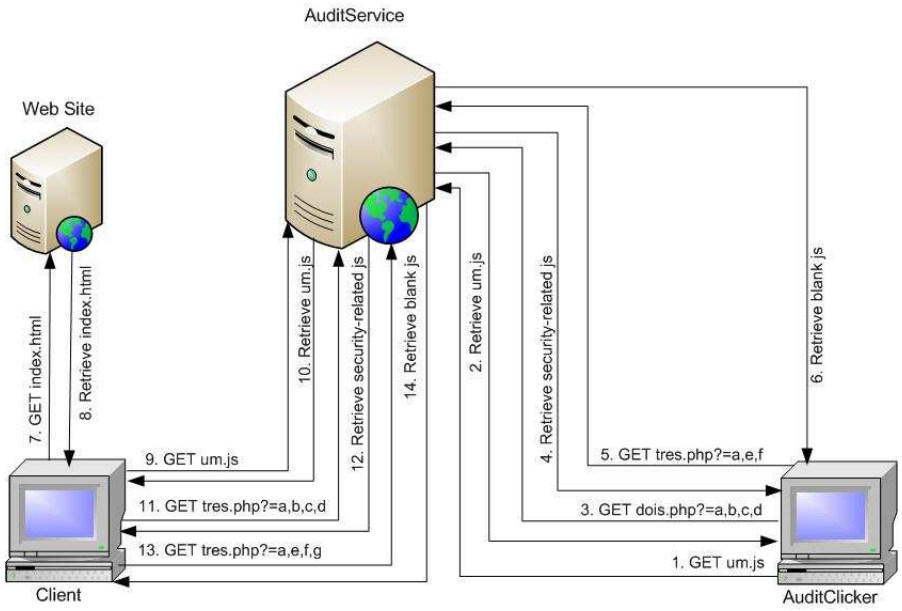

Figura 4.5.1. - Cenário do Teste de Cliques Provenientes de Ferramentas de Clique Automático.

No *AuditProfiler*, os dois grupos de tráfego estão associados a contratos diferentes. Um contrato é constituído por um expositor e um anunciante. No caso do tráfego real, o expositor chama-se *Blank*, porque não existe e o anunciante é o sítio *Web*. Para o tráfego gerado pelos *AuditClick* foi criado um contrato, em que o expositor se chama *AuditPublisher* e o anunciante se chama *AuditAdvertiser*. Os cliques identificam o contrato a que pertencem através dos *referers* que transmitem. O *referer* do anunciante é introduzido como cabeçalho do pedido HTTP e o referer do expositor é passado como parâmetro do GET.

O *AuditClick* foi configurado para gerar 1.000 cliques a uma cadência aproximada da cadência dos cliques provenientes do sítio *Web*, que é de aproximadamente 7.000 por dia o que equivale a quase 5 por minuto. O *AuditClick* é iniciado imediatamente após a activação da captura. A captura é terminada imediatamente após o último (1.000º) clique gerado pelo *AuditClick*. A captura contém 2.328 cliques, 1.000 do *AuditClick* e 1.328 do sítio *Web*. Uma vez capturado o tráfego, executa-se o ficheiro de reconhecimento.

Pelo facto da captura realizado conter dois tráfegos pertencentes a contratos diferentes, os resultados podem ser analisados em três vertentes: 1,2) análise de cada um dos tráfegos isoladamente; 3) análise do tráfego global. O tráfego proveniente do sítio *Web* já foi analisado no teste anterior, daí serem interessantes para este teste, apenas as análises do tráfego do *AuditClick* (A) e do tráfego global (G). Pelas estatísticas da tabela 4.5.1., confirma-se como o metade dos cliques do *AuditClick* imitam o browser *Firefox 2* e a outra metade o *Opera 9* (500 cliques de cada). Este facto prova que a imitação, até nos casos de menor precisão, é suficiente para ser correctamente detectada. A pontuação máxima dos cliques tem vários níveis reflectindo os vários graus de precisão da imitação.

Tabela 4.5.1. - Estatísticas do reconhecimento do Teste de Cliques Provenientes de Ferramentas de Clique Automático. Análises separadas do tráfego do AuditClick (A) e do tráfego global (G).

Total Clicks (A): 1000
Average MaxScore (A): 1.4812999953032
Total Clicks (G): 2328
Average MaxScore (G): 1.7549398574735

Test	Count A	Perc A	Count G	Perc G
Browser(MaxScore) = Firefox 2	500	50	742	31.8728522337
Browser(MaxScore) = Firefox 3	0	0	55	2.36254295533
Browser(MaxScore) = MSIE 5	0	0	4	0.171821305842
Browser(MaxScore) = MSIE 6	0	0	372	15.9793814433
Browser(MaxScore) = MSIE 7	0	0	627	26.9329896907
Browser(MaxScore) = Opera 9	500	50	514	22.0790378007
Browser(MaxScore) = Safari 5	0	0	14	0.601374570447
MaxScore < 2	992	99.2	1256	53.9518900344
MaxScore < 1.9	934	93.4	1110	47.6804123711
MaxScore < 1.8	842	84.2	883	37.9295532646
MaxScore < 1.7	763	76.3	775	33.2903780069
MaxScore < 1.6	677	67.7	689	29.5962199313
MaxScore < 1.5	476	47.6	480	20.618556701
MaxScore < 1.4	305	30.5	309	13.2731958763
MaxScore < 1.3	156	15.6	159	6.82989690722
MaxScore < 1.2	42	4.2	43	1.8470790378
MaxScore < 1.1	0	0	1	0.0429553264605

MaxScore < 1	0	0	0	0
MaxScore < 0.5	0	0	0	0
Browser(MaxScore) != Browser(UA)	83	8.3	148	6.35738831615
Browser(MaxScore) Name != Browser(UA) Name	0	0	9	0.386597938144

Existe uma diferença de quase três décimas de ponto entre a pontuação média do tráfego do *AuditClick* e a pontuação média do tráfego global, o que significa que se pode através da pontuação distinguir o tráfego automático do tráfego normal. Os cliques automáticos representam 42,96% do tráfego global. Na prática, segundo o método aplicado, só são classificados como inválidos os cliques que não têm uma correspondência mínima com nenhum dos perfis do sistema de reconhecimento. A correspondência é medida pela pontuação. É necessário estabelecer um limiar (*Threshold*) de pontuação mínima, que distinga o máximo de cliques inválidos, minimizando os falsos positivos. Um falso positivo é um clique válido, classificado como inválido. Na tabela 5.5.x procura-se um valor para esse limiar, que satisfaça as condições exigidas. A percentagem de falsos positivos (coluna '(Count G – Count A / Total G (%)') deve ser inferior a 1%. O limiar 1,7 tem 0,52% de falsos positivos e detecta 32,77% dos cliques inválidos presentes na captura, que corresponde a 76,3% dos cliques inválidos totais (763 / 1000). 1,7 seria o limiar mais equilibrado neste teste, para distinção entre cliques reais e cliques automáticos.

Tabela 4.5.2. - Limiar (*Threshold*) de pontuação. Cliques com pontuação inferior são classificados como inválidos. 'Count A' é o número de cliques do *AuditClick* com pontuação inferior a 'Threshold'. 'Count G' é o número de cliques globais (*AuditClick* + sítio *Web*). 'Total G' é o número total de cliques globais, 2.328.

Threshold	Count A	Count A / Total G (%)	Count G	(Count G - Count A) / Total G (%)
2	992	42,61168385	1256	11,34020619
1,9	934	40,12027491	1110	7,560137457
1,8	842	36,16838488	883	1,761168385
1,7	763	32,77491409	775	0,515463918
1,6	677	29,08075601	689	0,515463918
1,5	476	20,4467354	480	0,171821306
1,4	305	13,10137457	309	0,171821306
1,3	156	6,701030928	159	0,128865979
1,2	42	1,804123711	43	0,042955326
1,1	0	0	1	0,042955326

O *AuditClick* foi configurado para preencher correctamente os parâmetros de verificação de integridade, o que pode ser comprovado pelos resultados da tabela 5.5.x. As falhas patentes no tráfego global provêm dos cliques do sítio real e têm uma distribuição aproximada à do teste anterior

Tabela 4.5.3.- Estatísticas de verificação de integridade, da informação devolvida pelo cliente, no Teste de Cliques Provenientes de Ferramentas de Clique Automático. Análises separadas do tráfego do *AuditClick* (A) e do tráfego global (G).

Test	Count A	Perc A	Count G	Perc G
tres.php Not Requested	0	0	21	0.90206185567
MC Validation Test = Failed	0	0	1	0.0429553264605
Repeated MC Test = Failed	0	0	10	0.429553264605
CRC(dois.php) = Failed	0	0	0	0
CRC(tres.php) = Failed	0	0	0	0

4.6. Conclusão

Os testes de cliques manuais e de cliques provenientes de sítio Web real têm o objectivo de perceber a eficácia do AuditProfiler no reconhecimento de browsers. O teste com a ferramenta automática de clique pretende determinar que critérios têm de ser tidos em conta na detecção de cliques fraudulentos. O reconhecimento de browsers tem um sucesso de 100% na primeira experiência e de 98,87% na segunda. As fragilidades reveladas pelos testes foram as seguintes:

- Insuficiências do tráfego reunido para a criação dos perfis:
    - Não contempla tráfego encaminhado por servidores proxy;
    - Não contém algumas variantes de algumas características HTTP;
    - Tráfego produzido por browsers menos comuns como Konqueror (layout engine KHTML) ou browsers de dispositivos móveis.
- *Magic Cookies* repetidas.

O problema do tráfego insuficiente pode ser resolvido capturando tráfego continuamente que vá actualizando os perfis dinamicamente, e usar fontes de tráfego heterogéneas, por forma a que os perfis reflictam toda a diversidade de browsers existentes. O problema das *Magic Cookies* repetidas resolve-se introduzindo, para além do tempo (data e hora), uma variável única a cada cliente, como por exemplo o endereço IP.

A detecção de cliques fraudulentos tem um sucesso de 76,3%. O teste com o *AuditClick* não pretende simular um cenário de cliques fraudulentos real. O objectivo do teste é abordar os critérios que devem orientar a definição de um limiar mínimo de pontuação, que distinga os cliques automáticos dos manuais, os inválidos dos válidos. Quanto menor for a pontuação de um clique, menor é a semelhança entre o clique e um browser. Daí surge o indício de um clique fraudulento. Por outro lado, o limiar definido não pode classificar cliques válidos como inválidos, gerando falsos positivos. A escolha de um limiar tem de se pautar pela maximização da detecção de cliques inválidos e minimização de falsos positivos.

Concluindo, na maioria dos casos, o sistema reconhece correctamente os *browsers*, sendo ainda capaz de detectar aplicações que não se assemelhem com *browsers* (pela pontuação baixa que os seus cliques obtêm). O método de detecção de cliques fraudulentos por reconhecimento de browsers, parte da premissa de que as aplicações que realizam cliques

fraudulentos, as ferramentas de clique automático, têm características distintas dos browsers. O método só funciona se se verificar essa premissa.

5. Conclusões

5.1. Observações Principais

O trabalho proposto tinha como objectivo provar o conceito de detecção de cliques fraudulentos por reconhecimento de *browsers*. O reconhecimento de *browsers* tinha de ser feito do lado servidor. Para isso exploraram-se o comportamento de rede dos *browsers*, nomadamente na comunicação HTTP e o seu suporte dos *standards* W3C. A informação HTTP revelou-se muita rica para a prova de conceito, porque contém uma heterogeneidade entre os *browsers* que reforça a sua distinção. A componente de standards consiste em enviar programas para o *browser* que testem a sua compatibilidade com certas funcionalidades.

A solução implementada realiza os dois tipos de inspecção ao *browser* que foram estabelecidos como objectivos, paralelamente. Inserindo uma linha de código HTML na página a auditar (reconhecer os *browsers* que lhe acedem) conseguimos desencadear uma interacção com o nosso servidor auditor, em segundo plano e sem intervenção do utilizador que navega na página auditada. Dessa interacção é reunida informação de dois tipos (HTTP e DOM) num *log* formatado por nós e próprio para o posterior processamento de reconhecimento. A informação HTTP é retirada das mensagens HTTP em que o cliente pede os programas *JavaScript,* que vão realizar os testes de compatibilidade DOM. Os resultados dos testes são transmitidos ao servidor nos pedidos subsequentes programados na interacção.

A análise multifacetada do comportamento do *browser* revelou-se frutífera e a sua implementação resultou numa prova de conceito válida.

5.2. Principais Resultados

Os resultados deste trabalho podem ser observados de duas perspectivas: 1) o reconhecimento de *browsers* em si mesmo; 2) o reconhecimento de browsers como técnica de detecção de cliques fraudulentos. O sucesso dos resultados do reconhecimento de *browsers* prova que a abordagem adoptada neste trabalho é válida. A abordagem adoptada foi a de fazer um levantamento sobre a forma distinta como os *browsers* aplicam e suportam os vários standards que regem a *Web*. Por existirem, de facto, várias diferenças na forma como os *browsers* comunicam e suportam as funcionalidades propostas nos *standards*, é viável um reconhecimento baseado nessas características. Na segunda perspectiva, pela limitação dos testes realizados não se pode afirmar plenamente que a técnica é válida para detecção de

cliques fraudulentos. Por não termos encontrado ferramentas de clique automático capazes de interpretar código *JavaScript*, não nos foi possível realizar uma experiência, em que fossem definidos critérios concretos que detectassem esses cliques e os assinalassem como inválidos ou fraudulentos. A questão que ficou por explorar é de como se chega à classificação de um clique automático como inválido. A resposta a esta questão foi parcialmente respondida na experiência com o *AuditClick*, ferramenta de clique automática concebida pela AuditMark, e passa por estabelecer uma pontuação mínima para a validação de um clique. A experiência com o *AuditClick* deu orientações de como determinar essa pontuação mínima e que critérios respeitar para a detecção de cliques automáticos, num tráfego misto (com cliques válidos). O princípio da técnica é o de que se os *browsers* são todos diferentes uns dos outros, então as ferramentas de clique automático também terão as suas diferenças. Se nós conhecermos profundamente os *browsers*, e nos aparecer um clique, cujo padrão não apresenta semelhança suficiente com nenhum dos perfis dos *browsers*, então pode-se afirmar que aquele clique não proveio de um *browser*. Se não proveio de um *browser*, temos um indício de um clique automático. O sistema de detecção de cliques fraudulentos não terá como técnica única o reconhecimento de *browsers*. Outras técnicas referidas no capítulo 2, deverão ser acrescentadas ao sistema. Cruzando a informação das várias técnicas pode-se classificar um clique quanto à sua validade com o grau de certeza exigido ao serviço de auditoria que se pretende prestar.

5.3. Perspectivas e Desenvolvimentos Futuros

O sistema de reconhecimento de *browsers* implementado é insensível à qualidade dos resultados que produz. Introduzindo mecanismos de inteligência artificial que usassem qualidade dos resultados como referência, poder-se-ia automatizar a escolha dos valores dos parâmetros de entrada do sistema. Neste caso, a qualidade de resultados é tanto mais alta quanto maior for a percentagem de *browsers* correctamente reconhecidos. Os parâmetros de entrada são as características consideradas para comparação e a respectiva pontuação. Neste trabalho as características foram escolhidas e pontuadas racionalmente na base de observações feitas. Implementando um sistema inteligente a escolha de características e pontuações seria decidida pelo sistema em função da qualidade dos resultados que produzissem.

O segundo aspecto a desenvolver é o da manutenção dos perfis. Numa situação de tráfego puramente proveniente de *browsers*, a média da pontuação de reconhecimento deveria ser igual à pontuação máxima da escala pontual. Embora o sistema tenha conseguido uma boa aproximação (1,97 de um máximo de 2), esse valor poderia ser melhorado, se o tráfego que serviu a criação dos perfis fosse mais abrangente. Futuramente, a base de dados deveria ser constantemente actualizada com tráfego heterogéneo e de várias fontes. Também no caso da criação de perfis poderia ser aplicada inteligência artificial, nomeadamente no que diz respeito à filtragem dos perfis e variantes.

Em desenvolvimentos futuros também se pode explorar o suporte dos *browsers* relativamente a outros *standards* para além do DOM, nomeadamente CSS, HTML e XHTML. Testes de compatibilidade com *plugins* próprios para *browsers* como *Adobe Flash Player*, *Adobe Director Project*, *RealPlayer*, *Windows Media Player* e *QuickTime*, também podem trazer certeza e reforçar o reconhecimento.

Referências

[1] *An Industry Survey Conducted by PricewaterhouseCoopers and Sponsored by the Interactive Advertising Bureau, "Internet Advertising Revenue Report - 2007 Full-Year Results".* Disponível em: http://www.iab.net/media/file/IAB_PwC_2007_full_year.pdf. Acesso em: Junho de 2008.

[2] *Usage share of web browsers*. Disponível em: http://en.wikipedia.org/wiki/Usage_share_of_web_browsers. Acesso em: Março de 2008.

[3] Fielding, et al., W3C & IETF. RFC 2616 - HTTP/1.1. Junho 1999.

[4] DOMReference – doctype. Disponível em: http://code.google.com/p/doctype/wiki/DOMReference. Acesso em: Junho de 2008.

[5] Click Fraud Index. Disponível em: http://clickforensics.com/Pages/click-fraud-index.asp. Acesso em: Junho de 2008.

[6] Neil Daswani, Michael Stoppelman, and the *Google Click Quality and Security Teams. The Anatomy of Clickbot.A.* 2007.

[7] Alexander Tuzhilin. *The Lane's Gifts v. Google Report*. 2007.

[8] Shreeraj Shah. *Browser Identification for web applications*. 2002.

[9] Simone Soubusta. *On Click Fraud*. 2008.

[10] José Carlos Ramalho. Anotação Estrutural de Documentos e sua Semântica. 2006.

[11] *Click fraud*. Disponível em: http://en.wikipedia.org/wiki/Invalid_click. Acesso em: Junho de 2008.

[12] *Google Help - Invalid click or impression.* Disponível em: https://www.google.com/adsense/support/bin/answer.py?answer=32740&topic=8526):. Acesso em: Junho de 2008.

[13] *browserrecon project - advanced web browser fingerprinting*. Disponível em: http://www.computec.ch/projekte/browserrecon/. Acesso em: Junho de 2008.