UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

DEVICE FINGERPRINTING TECHNIQUES (THREATS AND PROTECTIONS)

Vítor Manuel Guerreiro Bernardo

Work oriented by Professor Doctor Maria Dulce Pedroso Domingos

DISSERTATION

MASTER ON INFORMATION SECURITY

2015

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Professor Doctor Dulce Domingos for the continuous support in the making of this work, for her guidance, knowledge and endless patience. This work would never have been without her wise advice, insightful perspective and positive reinforcement. Although most of the interaction was done in occasional meetings, it was an honour to share thoughts and experience the cheerful nature of Professor Dulce.

Besides my advisor, I would like to thank Dr. Isabel Cruz, Secretary General of the Portuguese Data Protection Authority, for guidance and advice during the definition of the work subject. I would also like to thank Professor Doctor Filipa Calvão, President of the Portuguese Data Protection Authority, for understanding and making possible my attendance to the master courses during working hours. The objective of getting a degree in Information Security was for me to become a better professional. I'll try to compensate the opportunity that was given by bringing in the knowledge that I gathered throughout the master.

A special acknowledgment to Dr. Nick Nikiforakis, which kindly answered questions and provided guidance as well.

Last but not the least, I would like to thank my family: my wife and daughter, my parents and grandmother for supporting me spiritually throughout writing this thesis and my life in general.

Dedicated to Goreti and Leonor,

for your never-ending love and patience.

Resumo

A evolução das tecnologias da informação ao longo das últimas décadas repercutiu-se de forma decisiva na nossa sociedade. Estes desenvolvimentos trouxeram sobretudo a possibilidade de expansão da atividade de diversos setores, possibilitando novas abordagens até então impraticáveis ou demasiado dispendiosas para pôr em prática com a tecnologia existente.

O setor comercial, em particular, viu o seu paradigma de negócio dramaticamente alterado com a introdução das novas tecnologias. O aperfeiçoamento dos serviços de logística e distribuição (também eles resultantes, em grande medida, de avanços tecnológicos), associado às novas tecnologias da informação permitiram ao setor comercial evoluir do conceito tradicional para novos modelos. O surgimento de redes de lojas associadas a uma determinada marca ou distribuidor, só foi possível graças a uma infraestrutura de comunicação que veio permitir a gestão de várias células dispersas como um único organismo. Por outro lado, a vulgarização de meios de comunicação como o telefone ou a Internet, permitiu às organizações chegarem junto dos clientes através de novos canais e tornou possível a desmaterialização dos pontos de venda e redução de custos com infraestruturas e pessoal.

Apesar das vantagens subjacentes aos novos modelos comerciais, o distanciamento entre comerciante e cliente, caraterística comum nas novas abordagens, trouxe um novo problema para o vendedor – este deixou de conhecer os seus clientes. Com efeito, esse distanciamento relativamente aos consumidores exigiu que os comerciantes idealizassem novas formas de antecipar as próximas compras ou de sugerir produtos eventualmente apelativos para um determinado consumidor.

Atualmente, quando o proprietário de uma superfície comercial pretende estudar os hábitos de consumo dos seus clientes, para fins de ajustamento da sua oferta à procura, necessita de agregar os registos de compras por cliente. O método mais comum para recolher esta informação baseia-se nos cartões de cliente. Mediante o preenchimento de um formulário com os dados pessoais do cliente e perante termos bem explícitos em que este seja devidamente informado do tratamento a que os seus dados serão submetidos, o cliente beneficiará de descontos ou outras vantagens nessa superfície. Desta forma o comerciante vê carregadas no seu sistema as aquisições afetas a um indivíduo, podendo assim aferir qual o agregado familiar, situação económica e interesses do titular dos dados.

Este tratamento de dados, ainda que possa parecer intrusivo, é legítimo desde que o cliente esteja devidamente informado relativamente ao tratamento que será feito aos seus dados e tenha dado autorização expressa para o efeito. O que seria considerado ilegítimo e intrusivo seria se cada cliente tivesse as suas compras secretamente registadas pela loja e associadas à sua pessoa (ou a um perfil com um conjunto de caraterística que o definissem) tendo efetuado o pagamento em numerário e sem que tivesse apresentado qualquer cartão de cliente. Um cenário ainda mais flagrante seria, por exemplo, uma situação em que fosse efetuado igual registo da entrada dos

visitantes na superfície comercial e eventuais consultas de artigos, apesar destes não terem efetuado qualquer compra.

Qualquer um dos tratamentos de dados supracitados configuraria um atropelo à privacidade e liberdade dos indivíduos. A ser registada de forma manual, a quantidade de informação a tratar tornaria este tratamento inexequível, mas as tecnologias da informação tornam isto possível, uma vez que a informação é tratada de forma automatizada pelos sistemas. O registo de informação não é problema para os sistemas computacionais, a única questão reside na associação dos dados ao indivíduo - *profiling*. É aqui que as técnicas de *device fingerprinting* na web permitem ultrapassar este problema.

À singularização[1] de um determinado dispositivo pela agregação de um conjunto de caraterísticas (físicas e/ou lógicas) desse sistema dá-se o nome de *device fingerprinting*.

Com base no princípio de que cada dispositivo possui um conjunto de caraterísticas, físicas e/ou lógicas, que o distinguem dos demais é possível criar padrões associados a comportamentos que são exteriorizados pelo dispositivo.

Padrões associados à componente de *hardware* baseiam-se no princípio de que pequenas diferenças físicas das componentes resultantes do processo de fabrico se repercutem no funcionamento dos dispositivos e são passíveis de serem registadas. Podem estar relacionados, por exemplo, com as características do sinal de redes sem fios transmitido por uma placa de rede (tal como apresentado por Speers, Goodspeed, Jenkins, Shapiro e Bratus (2014)) e Neumann, Heen e Onno (2012), com o sinal transmitido por um telemóvel (Hasse, Gloe e Beck (2013)) ou ainda com os ruídos introduzidos numa transmissão de som (por microfones e speakers, como explorado no trabalho de Das, Borisov & Caesar (2014)).

Este tipo de *fingerprinting* exige o processamento de sinais de rádio ou de outros tipos de radiação eletromagnética o que pode significar um esforço considerável do ponto de vista do tratamento dos dados (nomeadamente a remoção de ruídos e tolerância a perdas de sinal), para produzir informação passível de ser utilizada.

O *fingerprinting* que assenta sobre as caraterísticas lógicas do equipamento é, tipicamente, mais simples e viável já que, por norma, se baseia em informações que o sistema do cliente comunica (ou revela inadvertidamente) a um servidor.

O presente trabalho centra-se no estudo das técnicas de *web-based fingerprinting*, um tipo de *fingerprinting* especificamente dirigido aos navegadores web que permite extrair suficientes informações de sistema para criar uma assinatura deste.

[1] Entende-se por singularização a redução de um universo de indivíduos a um único (que pode ser identificável ou não) pelas caraterísticas exclusivas deste.

O processo passa pelo acesso do cliente, muitas vezes inadvertidamente, a uma página web que usa código de *fingerprinting*. O código de *fingerprinting* é tipicamente desenvolvido em linguagens executadas do lado do cliente (*cliente-side scripting*). Estas permitem efetuar chamadas a interfaces (Application Programming Interface, ou API) que acedem a informações de configuração do sistema operativo ou do próprio browser do cliente e as comunicam, de forma dinâmica, ao servidor.

Como qualquer tipo de tecnologia, o *web-based fingerprinting* pode ser utilizado para finalidades legítimas ou não. Na sua aplicação mais perniciosa permite agregar a informação que um dispositivo consultou em diferentes momentos, resultando daí um comportamento discriminatório para com o utilizador do dispositivo.

Talvez a aplicação mais comum seja a de criação de perfis para fins publicitários - *behavioural advertising*. Neste caso as páginas consultadas contêm código de *fingerprinting* que envia informação para outros sites (os chamados *third-party web-bugs*), que permite a parceiros do site visitado definirem perfis de utilizador para campanhas de publicidade. Um outro exemplo, uma loja virtual que aumenta os valores de um produto quando deteta que o mesmo dispositivo já acedeu à informação daquele item anteriormente.

Um dos piores cenários que envolve *web-based fingerprinting* é quando este permite identificar sistemas desatualizados como primeiro passo para instalação de software malicioso, como é o caso do *BlackHole Exploit Kit*[2].

É importante referir que o *web-based fingerprinting* pode ter aplicações legítimas, nomeadamente quando é usado em websites como forma de verificar se o dispositivo que um utilizador usou para se autenticar é diferente dos habitualmente utilizados, podendo assim exigir uma dupla autenticação como forma de prevenir ataques com credenciais roubadas.

Com as recentes questões de privacidade endereçadas na Diretiva 2009/136/EC do Parlamento Europeu e do Conselho relativamente à utilização de *cookies* sem consentimento expresso dos utilizadores, os administradores de websites podem ser tentados a enveredar para uma tecnologia não regulamentada, que apresente resultados semelhantes aos das cookies – o *device fingerprinting*.

Páginas web como o *Panopticlick* (https://panopticlick.eff.org/) mostram aos utilizadores quão únicas são as propriedades dos seus *browsers*, alertando-os assim para a eventual exposição ao *fingerprinting*.

---

[2] Descrição disponível sobre o *Blackhole Exploit Kit* disponível em https://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit/

Este trabalho tem como principal objetivo a identificação, classificação e estudo das técnicas de *web-based fingerprinting* atualmente conhecidas e a identificação das ameaças que estas apresentam à privacidade e segurança dos utilizadores.

A sua principal contribuição será a definição de uma taxonomia associada a estas técnicas, que permitirá catalogá-las com base no tipo de informação que permitem recolher. Cada técnica será avaliada quanto às vantagens e desvantagens da sua aplicação.

Finalmente, será feita uma reflexão sobre possíveis medidas de mitigação ou anulação do *fingerprinting* (propostas na literatura ou decorrentes do presente trabalho) e sobre a legalidade deste tipo de tratamento de dados à luz do atual quadro legal.

Pretende-se que este estudo represente um contributo válido para outros trabalhos na temática do *web-based fingerprinting* e que seja, simultaneamente, um meio de divulgação dos riscos para a privacidade que este tipo de processamento de dados encerra.

Palavras-chave: device, web-based, browser, *fingerprinting*, deteção

Abstract

The present work focuses on the study of *device fingerprinting* techniques currently used in websites to single out the visitor's devices.

Device fingerprinting is based on the principle that each device holds a set of physical and/or logical properties that might be used to discriminate it from the whole. Whether it's small physical differences in the device's components, stemming from the manufacturing process, that result in slightly different behaviors (Speers, Goodspeed, Jenkins, Shapiro & Bratus and e Das, Borisov & Caesar, 2014), the clock deviation of a client's device internal clock and the clock of a server (Kohno, Broido, & Claffy, 2005) or the set of fonts and browser plugins installed on a system (Eckersley, 2010), all of them allow a device to be profiled.

The *device fingerprinting* can be used for legitimate purposes, namely to verify if the device used in a user's authentication belongs to the set of devices usually used. If not, the server can request a second authentication, preventing attacks with stolen credentials.

The recent privacy issues stated in European Parliament & Council's Directive 2002/58/EC, addressed to the use of cookies without user expressed consent, might drive website administrators to deviate into non-regulated technologies that are able to bring them the same results that cookies did – such as *device fingerprinting*.

Websites such as *Panopticlick* (https://panopticlick.eff.org/) show the users how unique their browser's properties are, warning them about the dangers of exposition to *fingerprinting*.

The main contribution of the current work is to identify, categorize and study all *web-based fingerprinting* techniques currently known and identify the threats to security and privacy of users they impose. The advantages and disadvantages of each technique will be evaluated. Lastly, a reflection will be done on possible mitigation measures or cancellation of *fingerprinting* (proposed in the literature or resulting from this work) and on the legality of such processing of data in the light of the current legal framework.

Hopefully, this work will result in a useful baseline for further works on the *device fingerprinting* and for public awareness of the threats to privacy and computer security involved.

Keywords: device, web-based, fingerprint, plugin, detection

Contents

Image Reference

Table Reference

1 Introduction

The Austrian writer Ernst Fischer once said that "as machines become more and more efficient and perfect, so it will become clear that imperfection is the greatness of man".

One can argue that machines may never be considered perfect, as they are products of "imperfect" human minds and therefore, will always be flawed somehow. Even though, most people would agree that the work done by a (well designed) machine, is expected to be consistently similar throughout its routine[3]. Therefore, industrially manufactured goods are commonly seen as indistinguishable from each other because they went through the same processes and transformations. Maybe the best example of this belief is the rising trend and demand on certain[4] hand-crafted products over industrially manufactured products, because the former have small differences between each other, introduced by the "imperfection" of a human action, which make them unique and therefore, more "organic".

Of course, not all industries are affected by these trends. From the automotive or information technology manufacturers, for instance, customers expect exactly the same level of performance and behaviour, between different units of the same product model. In this scenario, a deviation from the norm is rarely well received, as it gives away the impression of lack of control over the manufacturing process.

However, studies have shown that even goods produced by carefully calibrated machinery, such as computer processors, microphones, speakers, or computer network cards, have microscopic imperfections introduced during the manufacturing process. Changes in the temperature of the facility, atmospheric pressure, humidity or the variation in the temperature of a soldering machine – all of these can contribute to minor, infinitesimal differences between products.

Luckily these differences are negligible for the majority of the users, as they don't result in visible differences in the product's behaviour. However, if the component externalizes its action somehow, whether because it transmits information, emits noise, heat or electromagnetic radiation, then an external agent can gather such readings throughout time.

---

[3] We are obviously ruling out faults introduced by users or external agents, failures due to erosion and extensive wear and other external influences that could change the ideal working scenario.

[4] Products like ceramics, music instruments, jewels, clothes, decoration, etc.

Device fingerprinting is based on the principle that no two components are absolutely equal and that it is possible to create profiles of the emanations patterns sent or leaked from the devices, as long as the externalizations are repetitive through time.

A particular case of *device fingerprinting*, the *web-based fingerprinting*, allows a website owner to uniquely identify a particular client's device based on a set of information elements transmitted by the browser used to access the Internet. The present work focuses on the identification and study of *web-based fingerprinting* techniques.

The unique identification of a device allows a website owner to track that device's accesses throughout time, in an almost invisible way. In fact, even if the user is aware of privacy issues and takes precautions, whether by deleting cookies, blocking all cookies or using a browser in "private mode", he will still be "fingerprintable". This makes the use of *fingerprinting* far more upsetting than simple cookies.

In most cases, *web-based fingerprinting* is used to track user's activity in sites and bind a device to a user profile (together with its preferences, tastes and interests). It's easy to guess that advertising companies are obviously interested in this kind of information, as it allows them to address focused publicity through the partner site (the page consulted by the user).

As stated[5] by the Article 29 Data Protection Working Party, a European Union advisor on data protection, in the Opinion 9/2014 (Article 29 Data Protection Working Party, 2014) on the application of Directive 2002/58/EC (European Parliament and the Council of the European Union, 2002) to *device fingerprinting*:

> *(...) fingerprint provides the ability to distinguish one device from another and can be used as a covert alternative for cookies to track internet behaviour over time. As a result, an individual may be associated, and therefore identified, or made identifiable, by that device fingerprint. (...)*
>
> *The data protection risks of device fingerprinting are increased by the fact that the unique set of information elements is not only available to the website publisher, but also to many other third parties.*

There are, however, legitimate uses for such techniques. A website that keeps track of the devices commonly used by a client to access it, is able to identify a suspect device trying to access that user account. In this case, the system could trigger a series of questions to make sure the user is legitimate or use a kind of multi-channel authentication. Other legitimate use could be the adapting of the user interface to the characteristics of the device accessing it.

[5] Article 29 Data Protection Working Party (2014)

In both cases, explicit consent from the user would be required in order to apply the *fingerprinting*.

Based on the principle that no two components are absolutely equal, it is possible to create patterns of the emanations sent or leaked from the devices

Web-based fingerprinting relies in the device's operating system properties, installed software and other logical configurations in order to get a unique signature from the device - rather than trying to infer patterns from the behaviour of hardware. This has several advantages. First it is quite cheap to implement, as it does not require special equipment or a specific scenario to be put into practise. Second, the hardware *fingerprinting* typically requires the use of metrics or existing patterns in order to compare the information being processed – *web-based fingerprinting* does not require any information in advance, relying only in the one collected.

The way it works is as follows: a client accesses a web page that has *fingerprinting* code, the client's browser executes a portion of a client-side scripting language (generally JavaScript/jQuery) that makes API calls to the operating system and to the browsers preferences, the browser sends the information dynamically to the server, without any involvement from the user.

The following paragraphs will present the structure of this work, as well as the objectives pursued and the methods used in the study of *web-based fingerprinting*.

1.1 Motivation

Web-based fingerprinting is still a fairly new concept. In 2010, Peter Eckersley published the work "*How Unique Is Your Web Browser?*"[6] and paved the way for other researchers to build upon. In another work (Nikiforakis, Joosen, & Livshits, 2014), a group of researchers stated that from a universe of the top 10,000 most popular sites (ranked by the site Alexa) only 40 were confirmed as having *fingerprinting* code at the time. This proportion can be interpreted in two different ways: *fingerprinting* is not appealing to site owners or *fingerprinting* capabilities are still unclear or unknown to site owners. The author of the current work believes that the *fingerprinting* techniques are still not widespread, making this an interesting subject to be explored at the present time.

[6] See Eckersley (2010)

With the limitations that the European Commission is trying to impose to the use of "unconsented" cookies[7] many site owners will have to turn for a different solution or risk to face penalties. *Fingerprinting* presents itself as a first contender for this replacement: it can be implemented in multiple different ways[8], it's highly stealthy and, at first glance, it seems that there is no regulation addressing it. This last section will be clarified throughout this work.

Aside from a FCUL student, the author of the present work is also a member of the Portuguese Data Protection Authority (DPA) for which privacy issues are an important topic. Not only the Portuguese DPA, but also other European authorities agree that *device fingerprinting* might stand as one of the biggest threats to privacy on the Internet.

Having that said, the main motivation for this work was the study of the potentialities and risks imposed by this technology hoping that this could lay work for others.

1.2 Other types of *device fingerprinting*

It's important to stretch that *device fingerprinting* can take many different approaches, depending on the nature of the data collection and the channel used for data gathering.

Working devices externalize their activity in multiple different forms, whether it is by sending communication signals (in electromagnetic form, acoustic waves or other), generating heat, producing movement, emitting noise, etc. As long as there is a way to capture and register the properties of said externalizations and if these are continuous (i.e. not incidental) or happen as a response to an action, then the definition of a pattern is possible. With enough detail of the captured properties, the patterns can be used to create individual signatures – *device fingerprinting*.

The *fingerprinting* of computing devices is commonly based on the communications they establish with other devices. While *web-based fingerprinting* relies mostly in the information at the upmost layer of the Internet Protocol Suite (i.e. Application layer), other approaches explore lower layers of the protocol suite.

Speers, Goodspeed, Jenkins, Shapiro and Bratus (2014), for instance, explored the characteristics of the transmitted wireless signal of a network card and described a technique based on a stimulus frame with a non-standard physical-layer header that was transmitted and

[7] With the so called "ePrivacy Directive" (Directive 2002/58/EC amended by Directive 2009/136/EC). The Article 5(3) of this directive states that "*the storing of information, or the gaining of access to information already stored, in the terminal equipment of a subscriber or user*"

[8] See the Section 1.2 shows other types of *device fingerprinting*

the target's response or lack thereof was recorded. The authors showed that it was possible to distinguish different radio chipsets by which type of stimulus packets they were able to receive.

Another approach on *device fingerprinting* based on network card behaviour was showed by Neumann, Heen, and Onno (2012) who described a *fingerprinting* method applicable on encrypted 802.11 traffic. It consisted in monitoring 802.11 frames using a standard wireless card in monitoring mode on a specified 802.11 channel. Because the IEEE 802.11 standard is very loosely implemented in many wireless cards, a combination of parameters such as transmission rate, frame size, medium access time or transmission time can allow a creation of an individual signature.

The patterns of web traffic of a mobile device when it synchronizes its apps were also explored by Stöber, Frank, Schmitt and Martinovic (2013), who showed how a *fingerprint* can be made from characteristic mobile device traffic patterns, especially in the time domain and in the volume of transmitted data.

It should be noted that the examples presented are applied in different scenarios (for instance, some require the *fingerprinting* agent to be able to recognize the traffic and others require great proximity with the targeted device).

1.3 Scope of the work

This work focuses only in *web-based fingerprinting*, meaning that only data which can be collected from the interaction of the client browser with the web-server(s) during the browsing activity is considered.

Beyond the scope of this work are the types of *device fingerprinting* that exploit the following features: physical-layer identification using specific equipment (e.g. collection of GSM or WIFI signals during Internet access, explore different acoustic features of a component); network traffic patterns (e.g. *fingerprinting* characteristic traffic generated by applications, exploring LAN or DNS traffic patterns); biometric patterns (e.g. typing or mouse moving patterns of the user) and any other type of *fingerprinting* that is not web-based.

1.4 Related Work

This section will show the work currently existing to date that verses about *web-based fingerprinting* applied to browser software. Since the purpose of this work is not to develop new *fingerprinting* techniques but to study the existing ones, the works referenced on this chapter will report only to analysis of techniques as well.

As far as could be found in the existing literature, none of the previous works studied the techniques with a holistic approach. By holistic approach it is meant that an effort was made to gather information about all currently techniques of *web-based fingerprinting*, subjecting them to the same analysis in order to achieve comparable and measurable results.

The study done in the study "Cookieless Monster: Exploring the Ecosystem of Web-based *Device fingerprinting*" (Nikiforakis, Kapravelos, Joosen, Kruegel, Piessens, & Vigna, 2013) is the one that shares most resemblance with the present work. In this document the authors presented some of the practices of device identification through the *web-based fingerprinting* techniques available by then and measured the adoption of *fingerprinting* on the web. The authors also presented a taxonomy for the *fingerprinting* techniques found in "three large, commercial companies" [9] along with a popular site that displays what information can be gathered from the browser, *Panopticlick*[10].

Although very enlightening, the study is limited to the techniques that were used at the time by the three software manufacturers and the *Panopticlick* website. Therefore, other techniques like HTML5 canvas *fingerprinting* or exploitation of DNS leaks were not covered. It is worth mentioning that some techniques are more effective than others (as will be shown in Chapter 3) and that would explain the propensity to see those techniques applied more often. This shouldn't be a reason not to examine lesser used techniques, as they can be linked to other methods for increased efficiency. Therefore, the current study will aim to evaluate all known *web-based fingerprinting* techniques.

Furthermore, the "Cookieless Monster" work (Nikiforakis, et al. 2013) does not propose or evaluate any concrete measures to mitigate *web-based fingerprinting*.

Some of the same authors suggested (Nikiforakis, et al. 2014) a tailored browser (*PriVaricator*) which would make every visit appear different to a *fingerprinting* site, resulting in a different fingerprint for each visit. The work verses only about this proposed solution and evaluates it.

This technical measure will be evaluated in the current essay, together with other approaches.

The study presented "How unique is your web browser?" (Eckersley, 2010) is a well consolidated work and considered as one of the first to address *web-based fingerprinting*. However, and since it dates back to 2010, the work obviously lacks some of the most recent techniques.

Other works address specific *fingerprinting* technique leaving out comparisons with parallel technologies.

---

[9] *BlueCava*, *Iovation* and *ThreatMetrix*, see Nikiforakis, et al., 2013 for more details.

[10] Available in https://panopticlick.eff.org. *Panopticlick* was one of the first websites to show how web--based *fingerprinting* was possible and referred to the work of Eckersley, 2010.

With what has been stated above, it is fair to consider that there is a window of opportunity for a study that might congregate, evaluate and compare the existing techniques to date, while looking at them with a focus in security and privacy.

The result of said work would, hopefully, serve as a baseline for further works about *fingerprinting* and, at the same time, be used as an awareness tool.

1.5 Objectives

The main objective of the current work was to study and classify the *fingerprinting* techniques currently known in order to identify the threats these might impose to privacy and security.

A second objective was to reflect about possible measures to mitigate or cancel the *fingerprinting*, whether considering measures already proposed or suggesting new ones.

1.6 Methodology

Primarily, a research was made in order to gather all information available online regarding *device fingerprinting*. Information from multiple sources was collected, such as academic papers, technical information about *fingerprinting* products, *fingerprinting* source code and press articles. Efforts were made in order to get the latest techniques that were made public.

The techniques presented in this work were gathered from multiple sources, ranging from published papers, where such techniques were described in detail, to others found in websites used with a *fingerprinting* purpose – mostly websites that show how *web-based fingerprinting* is possible. Every technique presented will have a reference to the official work that spawned it and/or to the website where it can be seen being put into practice.

Secondly, the documentation collected was analysed and the techniques were disposed in a matrix, according to their characteristics, allowing the creation of the taxonomy that followed. The different techniques of *device fingerprinting* were evaluated regarding their threats to privacy and/or security of users and systems.

Finally, several approaches to avoid or mitigate the *fingerprinting*, from the client point of view, were considered and evaluated considering their efficiency and feasibility.

1.7 Document structure

This document is organized in the following way:

- Chapter 2 (*Web-based fingerprinting*) will show the way that websites perform the *web-based fingerprinting*. It will list the client-side scripting languages commonly used to retrieve client information. The final section of the chapter comprises a list of websites that depict *web-based fingerprinting* techniques;

- Chapter 3 (*Web-based fingerprinting* techniques) will contain the nuclear part of this work: the presentation of the techniques and the taxonomy. Here, the techniques will be classified by the level of threat they represent to the user's privacy and security;

- In Chapter 4 (Threats of *web-based fingerprinting*), the privacy issues that the fingerprint arises will be debated, as well as the main differences between traditional cookies and web-based fingerprinting. The security issues inherent to web-based fingerprinting will also be debated;

- Chapter 5 (Mitigations to *web-based fingerprinting*)contains the analysis of measures proposed in literature and suggestions of the present author, considering their efficiency and likeability to be used;

- Chapter 6 (Conclusion) shows the author's closing thoughts on the problems surrounding *web-based fingerprinting* and the challenges for the future of privacy and computer security. Also contains cues for future work on this subject.

2 *Web-based fingerprinting*

This chapter will firstly explain how the principle of *web-based fingerprinting* works. On the second part of the chapter the types of techniques will be described and evaluated.

2.1 How *web-based fingerprinting* works

The Image 1 shows how *web-based fingerprinting* works.

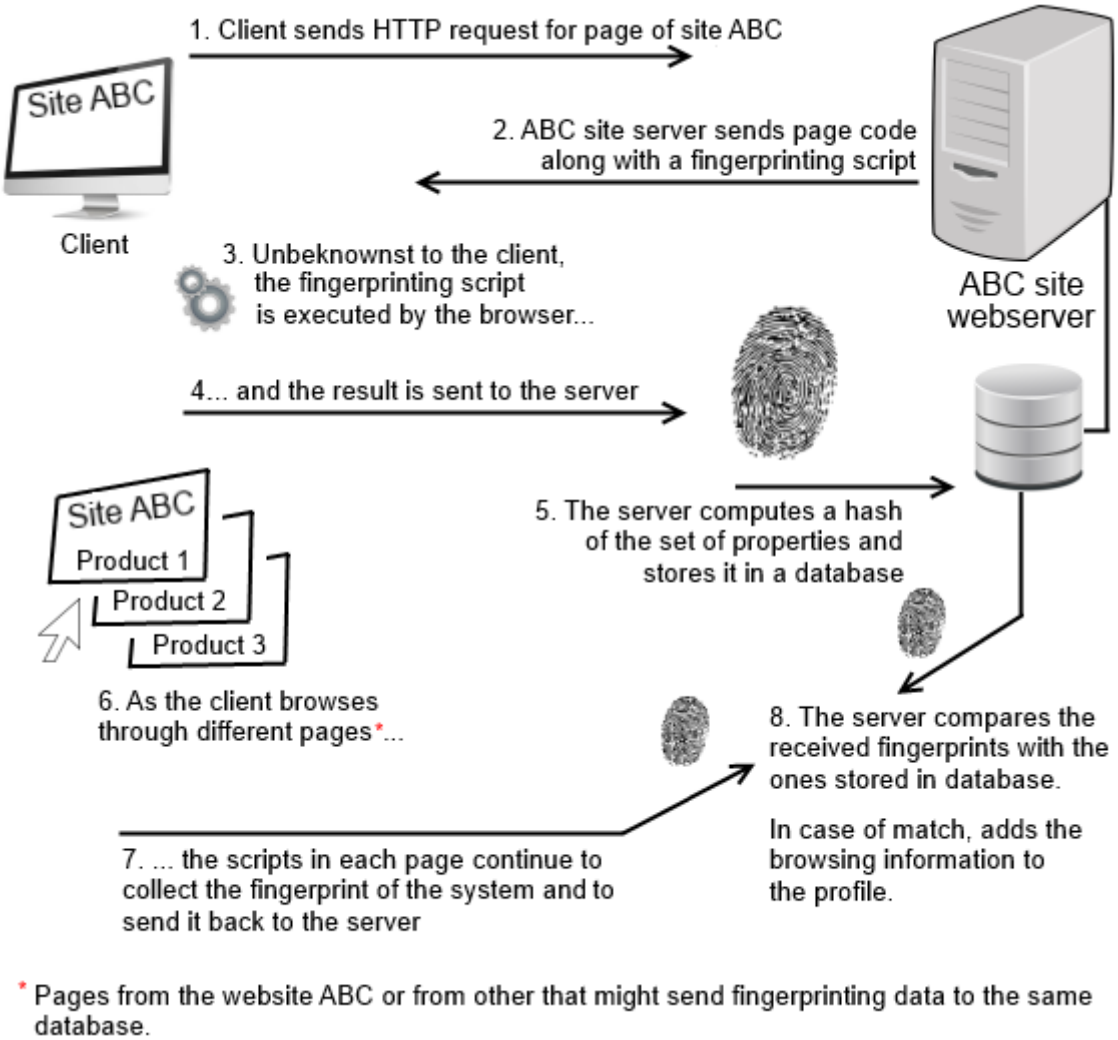

Figure 1- How web-based fingerprinting works (client-server only)

The Image 1 presents the several steps of the *fingerprinting* process. First the client's browser requests the server (through HTTP request) to send it the webpage code in order to render it for the user. In this request some *fingerprinting* information can already be collected from the user's system, namely the information that is sent in the User-Agent, a HTTP header field that the most popular browsers use to indicate the browser version and operating system.

In the HTTP response, the web server sends de page code that will enable the browser to render the webpage. Most *fingerprinting* techniques are based in client-side code execution because this technology allows the browser to make direct calls to the operating system or other machine configurations and send that data back to the webserver asynchronously (without interfering with the display and behaviour of the existing page). In practice, the client's browser communicates more than once with the web server for one simple page retrieval: first to request the page and then to send the data requested by the client-side script.

JavaScript is the commonly used language to achieve client-side scripting. Its popular library, jQuery, adds the asynchronous potential of the AJAX language, allowing the results of the processed scripts to be sent to the server, without the user's interference.

After the browser has processed the script and sent the result to the web server, the later computes a hash of the information received and stores it in database - this will be the fingerprint for this user's device.

Now, the website owner can track that device throughout the pages it consults, without the use of cookies, as long as they all contain the *fingerprinting* script.

It is important to note that the tracking can be done among different sites or domains, as long as all the websites share the same *fingerprinting* database. A more complex example of *fingerprinting* will be shown in Image 2, where the agent doing the tracking is not the website provider but a third-party agent. This scenario is described in 2013 Nikiforakis's work.

Companies dedicated to online advertisement are common nowadays. In order to make the advertisement process more efficient, advertising companies try to address to potential customers with products that these might find interesting/appealing. This can be done by creating a user profile, and *web-based fingerprinting* is all about that.

A site owner can agree with an advertisement company the deployment of a content from the later in the webpage, which visitors will retrieve unknowingly, containing a *fingerprinting* script.

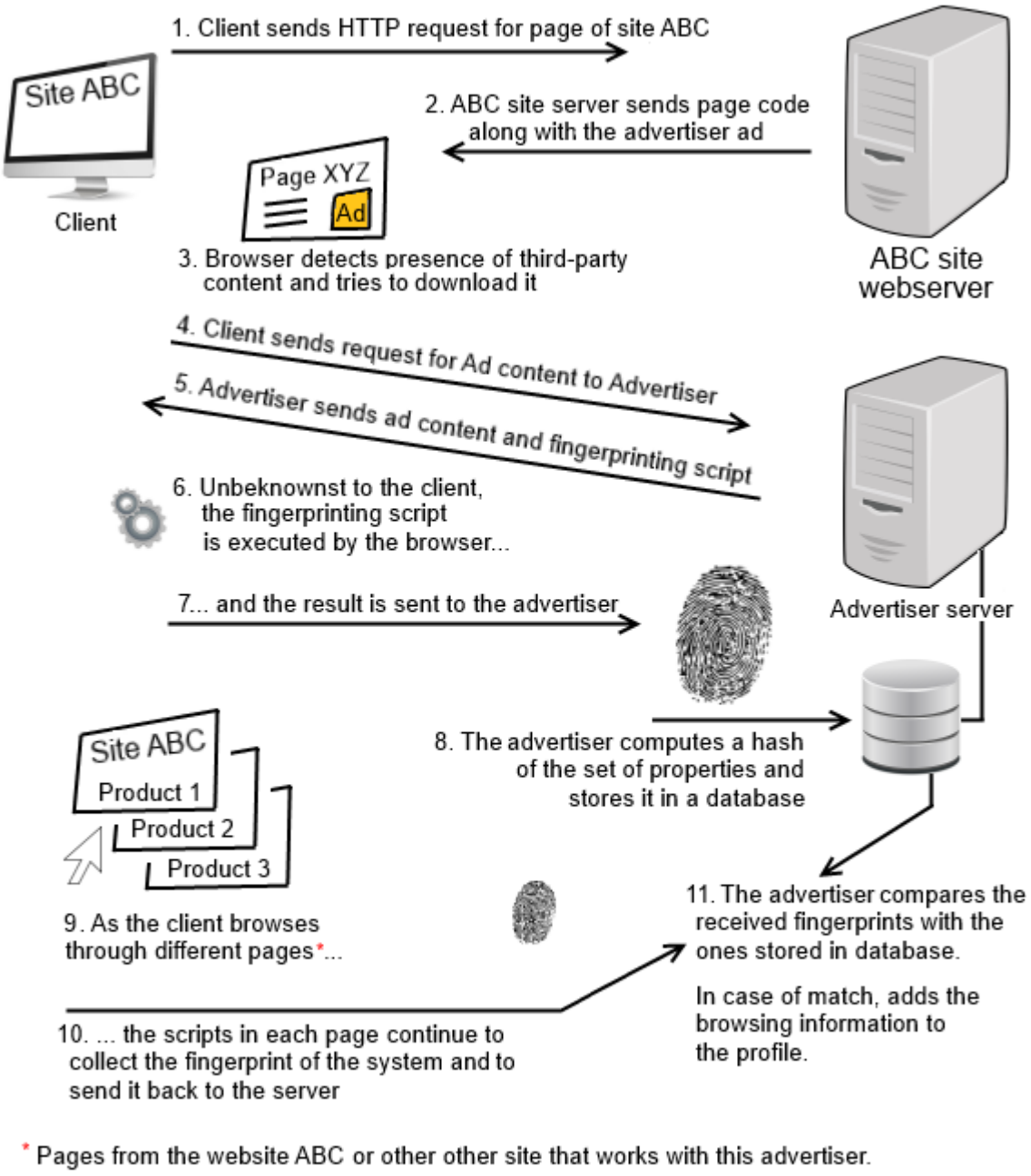

Figure 2- How web-based fingerprinting works (introducing the advertiser player)

This version shows a much more disturbing scenario: the *fingerprinting* agent can track users on multiple websites that he doesn't need to own, as long as the site owners agree to include the advertisement. This allows a much more intrusive profiling of the web users as different kinds of websites (with different subjects) provide a richer view of the individual. For example, the same device can be spotted browsing for running shoes in an online sport store and later accessing a page related with weight loss, both containing ads from the save advertise provider. Depending on the profiling algorithm used, this user could be profiled as someone with interests, for example in "weight loss", "sports" and "healthy food", allowing the advertiser to direct more specific ads to future visits of that user.

The risks to privacy inherent to profiling through *fingerprinting* will be discussed in Section 4.1 (Threats to privacy).

2.2 Browser and cross-*browser fingerprinting*

Web-based fingerprinting techniques can take two different approaches regarding which client-side set of features will be processed to extract a signature. One of the approaches relies on the distinctive features of the client browser and operating system (*browser fingerprinting*), the other approach is based only in non-browser features (cross-*browser fingerprinting*). The description of each one follows:

- ***Browser fingerprinting*** – This type of processing collects information from the client browser and the operating system. The purpose is to create a unique signature based on the information collected from both.

- ***Cross-Browser fingerprinting*** – Unlike the previous approach, this processing is not tied to a specific browser. Therefore, even if the client uses multiple browsers, the website is able to associate him/her with the previous profile. This approach requires system settings to be collected such as operating system version, CPU information, network interfaces information, number of processors, screen size, etc.
  From a *fingerprinter* standpoint, this kind of approach is more reliable, because it does not rely on the browser, but on the system.

For each of the technique described in Chapter 3, a reference will be made to which of the former types of *fingerprinting* is being performed. In some cases, a technique might perform both.

2.3 Technologies used in *web-based fingerprinting*

The following taxonomy aims at identifying techniques rather than technologies. In other words, a certain collection of information can be performed with different technologies. In the current context, "technology" refers to the means available for a website owner to collect information from client users, whether it's a web programming language or a communication protocol feature.

In order to avoid duplicating the same method, the technologies will be presented in the current chapter and referenced along the taxonomy whenever they can be used in a certain *fingerprinting* method.

The *fingerprinting* techniques present in the current taxonomy use at least one of the technologies presented as follows.

2.3.1 JavaScript

JavaScript is a dynamic programming language that allows the execution of code on the client-side. This technology allows a webpage embedded script to invoke browser and system properties from the client's device.

This language makes use of objects to refer to components of the client's system. Each object has a set of properties that allows the access or change of parameters. These properties can allow, for instance, a browser to indicate the existence of a plugin that might be used to execute a content of a webpage or assert that a certain font is present on the system. In general, these properties where built to improve the user's navigation experience, relying on an autonomous negotiation of settings between the browser and the webpage.

Although several issues have risen in the past, where browser vulnerabilities were exploited via JavaScript, the fact is that most users don't disable the JavaScript functionality[11]. This makes *fingerprinting* through JavaScript reliable since the majority of browsers in the market support this programming language.

2.3.2 jQuery

To be completely furtive, *fingerprinter*s rely on asynchronous web calls[12] where data can be sent to the webserver without interfering with the display and behaviour of the existing page, and without being noticed by the user.

AJAX is a group of Web development techniques that allows the creation of asynchronous web calls. A special library (jQuery) was designed to allow JavaScript to use AJAX's capabilities.

2.3.3 WebGL

WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D computer graphics and 2D graphics within any compatible web browser without the use

[11] In 2010 Yahoo analyzed the traffic to their sites and concluded that only 2% of the USA users had JavaScript disabled. See Zakas (2010).

[12] As opposed to the traditional synchronous web calls of HTML where an executing process would block all execution until that process was completed.

of plug-ins. When a browser supports WebGL, it can provide information about static WebGL parameters associated with the browser.

Note: It is important to stress that each vendor implements the compatibility with client-side script languages in their own way. In some cases, the call to a specific object or property has no results returned simply because the vendor did not implement them. *Fingerprinting* agents are aware of this and usually design their scripts in order to firstly check the browser's brand and then execute the calls to objects and properties that are known to be compatible.

2.3.4 Flash

This software platform gained worldwide notoriety during the nineties due to its potential to create vector graphics, animations and games.

Flash uses the ActionScript language that encompasses APIs to access operating system resources. Usually embedded in web pages, the Flash content is played by a browser plugin.

In what privacy and anonymity concerns, Flash plugins are known to be fragile. The Tor Project, responsible for the Tor Browser development, alerts the users about this issue in their site[13]. As stated in the FAQs section, "(p)lugins operate independently from Firefox and can perform activity on your computer that ruins your anonymity. This includes but is not limited to: completely disregarding proxy settings, querying your local IP address, and storing their own cookies." What the Tor Project team is trying to tell here is that Flash plugins can completely bypass some security measures that the browser might have implemented, allowing the leakage of system information to the webpages accessed. This is another important source of information to consider for *fingerprinting* purposes.

2.3.5 Silverlight

The application framework, Silverlight, was Microsoft's response to Adobe Flash. Aside from allowing the streaming of media, current versions of Silverlight also support multimedia, graphics, and animation, and give developers support for CLI languages

[13] Available at https://www.torproject.org/docs/faq.html.en#TBBFlash

and development tools[14]. In the same line of Flash, Silverlight plugins give no guarantee to the user that they will not completely bypass the browser control, rendering any privacy measures useless. This technology allows the retrieval of the client's OS Version, Processor Count, System Uptime, Time Zone, Installed Fonts, System and User Culture, Region and Language OS settings.

2.3.6 Java

One of the most popular object-oriented programming languages since the nineties, Java suffered serious blows to its image with the vulnerabilities discovered in 2012, with the Trojan Flashback[15], and in 2013 with multiple issues made public[16,17,18]. Following these events Apple released an update that removed the Java plugin from all its browsers[19] and Mozilla disabled Java in Firefox by default[20].

Nowadays, most browsers make visible alerts and require user's consent in order to execute Java Applets in the browser, making *fingerprinting* more difficult to perform. A Java Applet can access the following system information: operating system (name, architecture and version); java version and vendor; JVM (name, version and vendor); memory (available, free, maximum and total); CPU (available cores) and available network interfaces.

2.3.7 HTML5

HTML5 is a core technology mark-up language of the Internet used for structuring and presenting content for the World Wide Web. It includes detailed processing models to encourage more interoperable implementations and introduces mark-up and application programming interfaces (APIs) for complex web applications[21]. Both HTML5 and Flash include features for playing audio and video within web pages, and integrated vector graphics. However the former has gained notoriety over the years because it

---

[14] According to the Silverlight official site available at http://www.microsoft.com/silverlight/
[15] Trojan described in the article *Mac Flashback Exploiting Unpatched Java Vulnerability* (Brod, 2012).
[16] Article *A zero-day vulnerability was found in all versions of Java 7* (Constantin, 2013b).
[17] Article *The Red October cyberespionage attacks* (Constantin, 2013a).
[18] Article *Java's new "very high" security mode can't protect you from malware* (Goodwin, 2013a).
[19] Article *Apple removes Java from all OS X Web browsers* (Goodwin, 2012).
[20] Article *Firefox to block content based on Java, Reader, and Silverlight* (Goodwin, 2013b).
[21] As presented in the article *HTML5* from the W3C Working Group, available at http://www.w3.org/TR/2014/REC-html5-20141028/

doesn't require a plugin to be used (unlike Flash) and also because Apple openly stood against the inclusion of Flash plugins in their browsers[22].

The browser's level of compatibility with HTML5 features can be retrieved with JavaScript calls and allows to determine the browser family.

2.4 Websites for browser testing

A good starting point to evaluate the possible techniques of *web-based fingerprinting* can be found in a variety of websites that show the users what kind of information can be retrieved from their systems.

The following list contains the references to websites that perform *web-based fingerprinting*, or that use techniques that can be applied in *fingerprinting*.

- **Panopticlick** (available at https://panopticlick.eff.org)
  A research project by the Electronic Frontier Foundation, this website was one of the first to address the issues of *fingerprinting*, based on the 2010 pioneer work of Peter Eckersley. Appears to have one of the biggest known database of fingerprints (over 5,5 million signatures at the time of the writing) and uses JavaScript and Java for the data collection.

- **Noc.To** (available at http://noc.to)
  Includes an extensive array of techniques and technologies. Also makes use of Zombie Cookies to aware users about the risks of data that can be stored outside of the web browser's dedicated cookie storage, allowing the website to recreate deleted cookies.

- **BrowserLeaks.com** (available at https://www.browserleaks.com)
  A very complete display of *fingerprinting* techniques, from the typical JavaScript collection of DOM objects, Flash collection of fonts, HTML5 canvas, Java, etc.

- **PluginDetect Library** (available at http://www.pinlady.net/PluginDetect/All)
  Hosts a plugin detection library for download. The page shows the library in action by showing the browser's plugins.

[22] As officially stated by their former CEO Steve Jobs in 2010 in a public letter called *Thoughts on Flash*. Available at http://www.apple.com/hotnews/thoughts-on-flash/

- **Detect my Browser** (available at http://detectmybrowser.com)
  Website that uses multiple techniques.

- **latit.lab** (available at http://www.lalit.org/lab/javascript-css-font-detect)
  Shows how JavaScript code can be used to detect availability of a particular font in a browser.

- **MyHTTP.info** (available at http://myhttp.info)
  Tool that allows to see the HTTP request header.

- **DNS leak test** (available at https://www.dnsleaktest.com)
  Performs two modes of DNS leak testing (standard and extended) and explain how the technique works.

- **IPLeak.net** (available at https://ipleak.net)
  Performs the WebRTC-based attack to retrieve the local IP Address.

- **Darkwave Technologies**
  (available at http://www.darkwavetech.com/fingerprint/fingerprint_code.html)
  Contains a selection of downloadable *device fingerprinting* code for multiple techniques.

- **Cross-*browser fingerprinting* test 2.0**
  (available at http://fingerprint.pet-portal.eu)
  Uses multiple techniques, such as collection of screen resolution, font enumeration, plugin enumeration, list of mime-types, etc.

Summary

This chapter depicted how *web-based fingerprinting* works. It showed in detail what are the common communication steps and which actors can be involved in the process. Two scenarios were portrayed to help illustrate different approaches.

The main technologies to develop client-side scripts for *web-based fingerprinting* were listed and discussed. It's important to stress that these are not the only client-side scripting languages that can be used to retrieve information from the client's system, but are the commonly depicted in literature. Moreover, server-side scripting is also a type of technology that is able to collect data sent from the client side, although it lacks the possibility to make direct calls to the client's system APIs, limiting the *fingerprinting* capability. Since the present work aims to capture the

portrait of the *web-based fingerprinting* techniques, it was logical to follow what most of the literature refers as the main channel to reach client-side information, which is client-side scripting.

The last section listed some of the main websites that are publicly available (at the time of writing) and contain *web-based fingerprinting* tests. Many of them were built to aware web users to the threats that might lurk in webpages. Most of the examples that will be shown in the next chapter were collected from these websites.

3 *Web-based fingerprinting* techniques

This chapter will introduce and discuss the various methods of *web-based fingerprinting*.

Each technique will be described explaining the kind of information it allows to retrieve from the client's browser and what technologies can be used to perform it. A brief explanation will be presented as well as a few examples of the data sent to the *fingerprinting* agent.

In order to present some examples of the *fingerprinting* techniques, different browsers were used. The objective was to compare the different behaviours and outputs of each browser when faced with the same *fingerprinting* websites or scripts. This approach will allow to ascertain that different vendors show different approaches in the making of browser software.

The browsers used where: *Mozilla Firefox 38.0.1; Microsoft Internet Explorer 10.0.9200.17357*; *Google Chrome ver. 43.0.2357.81 m*; *Opera/9.80 (Windows NT 6.2; WOW64) Presto/2.12.388 Version/12.17*; *QtWeb Internet Browser 3.8.5 (build 108)*; *midori 0.5.10* and *Chromium Portable version 44.0.2383.0*.

The reason for choosing this specific set of browsers was to observe the behaviour and output of *fingerprinting* techniques both on widely used browser clients, such as *Internet Explorer*, *Google Chrome*, *Mozilla Firefox* and *Opera*, and less known software, such as *midori* or *QtWeb*.

By testing techniques with different browsers, it's possible to evaluate the degree of correctness and confidence that the technique provides: if a technique continually manages to collect the same number of elements from the client's environment independently of the browser used, then the technique proves to be reliable and thus, consistent enough to be used in *fingerprinting*. Bear in mind that we are not stating that different browsers should return the same information, but that the collection process across different browsers allows to retrieve the same number of properties.

As Nikiforakis stated in his work of 2014, "stability is a desirable property in a *fingerprinting* strategy".

Whenever possible, examples of different outputs from the different browsers will be included and compared. Due to size constrains, the output of some techniques will be limited to only one browser – specifically when the output is too lengthy to be presented multiple times.

All examples provided were retrieved by the author of the current work, using a Microsoft Windows 8 system and the browsers previously mentioned.

Furthermore, the advantages and disadvantages of each technique will be debated, from a *fingerprinter* standpoint. The advantages will represent strengths of that particular technique (e.g. capability to create a signature by itself, *cross-based fingerprinting* instead of *browser fingerprinting*). Inversely, the disadvantages will verse about weaknesses inherent to those techniques, from the *fingerprinter* standpoint (e.g. risk of incoherent signature collection throughout time, the need of a comparison database to map the collected information).

It is worth mentioning that the techniques presented are publicly widespread and were not created by the author of this document.

3.1 IP address

Hardly information that could lead to a creation of a trustworthy fingerprint, the IP address can still be useful to help tuning the *fingerprinting* process.

Consider a scenario where the *fingerprinter* identifies multiple equal signatures associated with different IP addresses, originating from different geographical regions, in a short period of time (ruling out the possibility of the client device being used while travelling). This could mean one of two things: first, the *fingerprinting* method might not be gathering enough information to produce different signatures, what would require the collection of additional data to increase diversity; or, second, the client is using some type of IP spoofing mechanism to hide its real IP address, in which case, the *fingerprinter* should disregard the IP address information.

The opposite scenario, multiple signatures associated to a single IP address, can be interpreted in one of the following ways: different devices behind the same NAT are accessing the page (thus, using the same IP address); the client is using a different browser or the client's system underwent through changes that affected the device signature. The client could be using a public proxy as well, sharing the same IP address for outside connections with other users (that are visiting the page as well).

Regardless of the particular scenario, and while not being a *fingerprinting* technique, the IP address can help to adjust and complete other methods of *fingerprinting*. IP addresses are sent in the Internet Layer of the TCP/IP model so, in theory, the website always receives this information.

Most server-side languages have methods to retrieve this information from the client HTTP request. On the client-side JSON (a language derived from JavaScript) can be used to retrieve this information.

The collection of the IP Address provides information to perform *cross-browser fingerprinting*.

Advantages

The IP address of the originator is sent for every HTTP request without need of any special scripting from the website owner.

Disadvantages

With the widespread use of NAT, for better management of addresses to companies and Internet Service Providers (ISPs), the tracking of a device through its IP address became unfeasible. Moreover, the ever-growing use of TOR networks and IP spoofing methods requires caution when associating an IP address to a device.

Variation #1: Local IP address (IPv4)

The website *IPLeak.net*[23] allows checking if the client browser uses the WebRTC API. WebRTC is a browser-to-browser application for voice calling, video chat, and P2P file sharing. The thing about WebRTC is that it implements STUN (Session Traversal Utilities for Nat)[24], a network protocol that allows an end host to discover its public IP address if it is located behind a NAT. The exploitation of this functionality allows a web page to know the local IP address of all network interfaces of the client device.

This is a good tool for the *fingerprinter*, because it provides additional information which allows him to better understand the collected data. With the client's local IP address it is possible to verify if two different signatures with the same external IP address correspond to two different devices behind a NAT.

According to the *IPLeak.net* website, *Mozilla Firefox* and *Google Chrome* are two browsers known to allow the use of the STUN protocol. Although the former provides a functionality to disable this feature, the later does not.

Of the tested browsers, *Mozilla Firefox*, *Google Chrome* and *Chromium* leaked the IP local address.

---

[23] Available at https://ipleak.net/

[24] When a client has discovered its external address, it can use it as a candidate for communicating with peers by sharing the external NAT address rather than the private address (which is, by definition, not reachable from peers on the public network). STUN protocol description available in http://www.voip-info.org/wiki/view/STUN

Variation #2: DNS leaks

Several websites[25] provide a way to test if a device leaks its DNS queries. When using services that provide anonymity (e.g. Tor network) or privacy connection (e.g. VPN), it is extremely important to route all traffic originating from the client's computer through the anonymity network. If any traffic leaks outside of the secure connection to the network, any adversary monitoring the client's traffic will be able to log his activity. When a DNS request is made to the default DNS servers (usually from the internet service provider), instead of the anonymous DNS servers assigned by the anonymity network, it is considered a "DNS leak". This is a major privacy threat since the anonymity of the connection is compromised.

Users should make sure that no traffic bypasses the anonymous/private connection.

Aside from the threats to anonymous/private connections, what websites such as "DNS leak test" show is that the retrieval of the client's DNS servers is a feasible process.

The technique is rather simple, the website generates a certain number of domain names and includes them in the code of the requested page. All of these domain names will have a common top and second level domain (such as EXAMPLE.COM), which is a real domain that belongs to the *fingerprinter*. The third-level domain will be a set of characters, programmatically generated, non-repeatable and will not correspond to any valid domain (the full domains will look like this: 7GF58KI.EXAMPLE.COM or KF5HT68.EXAMPLE.COM).

When the client browser finds the references to these domains in the page code it tries to resolve them, by querying its DNS servers. The DNS servers, in turn, don't have any cached registry for those domains (because they don't exist), so they will request for addresses to the domain authority (which is EXAMPLE.COM). Once the DNS servers (of EXAMPLE.COM) detect the call to the fake domains they forward the requests to the *fingerprinter*, together with the names of the DNS servers requesting them. Then, the *fingerprinter* can correlate the faux domains with the IP address that requested the page, and associate an array of DNS servers to that profile.

It is important to note that a list of DNS servers is not a reliable set of data to build a signature from. In fact, most ISPs (which provide the DNS service to the majority of end users) tend to use a wide array of DNS servers and a client might use a different set throughout the same session.

The best way to explore this feature would be to create a list of DNS servers associated to each of the device signatures. That list could be queried to keep track of the types of connections to Internet being used for this device.

[25] Websites like "DNS leak test", available at https://www.dnsleaktest.com/

3.2 HTTP Header fields

The HTTP protocol specification describes a series of header fields[26]. Header fields are colon-separated name-value pairs in clear-text string format, terminated by a carriage return (CR) and line feed (LF) character sequence, used to define the operating parameters of an HTTP transaction.

The information found in most of these fields doesn't differ much between systems, but some specific fields could give some information about the operating system and the web browser.

Because the header fields are actively sent by the browser upon the HTTP requests, any server-side language has the ability to retrieve the data, therefore requiring no client-side scripting.

Because HTTP header fields are a well-known source of information, many authors have mentioned them in their works. Eckersley (2010), Nikiforakis, et al. (2013), Acar, Juarez, Nikiforakis, Diaz, Gürses, Piessens and Preneel (2013) and Mowery, Bogenreif, Yilek and Shacham (2011) are examples of works that showed the HTTP header fields as a way to collect data.

Next, four of the HTTP Header fields that can be used to gather information about the client's system will be described - Accept, Accept-Encoding, Accept-Language and User-Agent.

Accept field

Allows the browser to inform the server about the Content-Types that are acceptable for the response – the media types that the browser understands and how well it understands them. This allows serving different versions of a document at the same URI.

This field provides browser related information and can contribute to the *browser fingerprinting*.

Examples

Several tests were made in the website *MyHTTP.info* (available at http://myhttp.info), using different browsers on the same device. Table 1 shows the collected values for the Accept field.

Browser	Accept value
Mozilla Firefox	`text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8`
Google Chrome	`text/html,application/xhtml+xml,application/xml;q=`

[26] See the HTTP protocol specification available in http://www.w3.org/Protocols/rfc2616/rfc2616.html

	`0.9,image/webp,*/*;q=0.8`
Microsoft IE[27]	`text/html, application/xhtml+xml, */*`

Table 1- Example of 3 different Accept field values

Accept-Encoding field

Since HTTP data is compressed before it is sent from the server, compliant browsers should announce to the server what methods they support before downloading the correct format. Browsers that do not support compliant compression methods will have to download uncompressed data.

This field provides browser related information and can contribute to the *browser fingerprinting*.

Examples

The following values for the Accept-Encoding (Table 2) were retrieved while testing with various browsers.

Browser	Accept-Encoding value
Mozilla Firefox	`gzip, deflate`
Google Chrome	`gzip, deflate, sdch`
QtWeb	`gzip`

Table 2- Example of 3 different Accept-Encoding field values

Accept-Language field

A webpage might have the same content available in several languages. The language can be selected manually by the user in the site's index page or left for the server to choose automatically, based on the preference indicated by the browser in this field.

This field may leak the user's nationality or mother tongue.

In the examples below it is clear that different browsers implement this property differently. Therefore, this property also contributes to *browser fingerprinting*.

[27] Abbreviation for *Internet Explorer*

Examples

Table 3 shows the different values collected for the Accept-Language field, for the same system.

Browser	Accept-Language value
QtWeb	`pt-PT,en,*`
Google Chrome	`en-US,en;q=0.8`
Microsoft IE	`pt-PT,pt;q=0.8,en-GB;q=0.5,en;q=0.3`

Table 3 - Example of 3 different Accept-Language field values

User-Agent field

According to the HTTP protocol specification:

> *The User-Agent request-header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. User agents SHOULD include this field with requests.*

This field can provide specific information about the browser brand and version. It also allows inferring the type of operating system, which makes it useful for *browser fingerprinting* and *cross-browser fingerprinting* at the same time.

Examples

The following different values depicted in Table 4 were retrieved for the User-Agent field.

Browser	User-Agent value
Mozilla Firefox	`Mozilla/5.0 (`**`Windows NT 6.2;`**` WOW64; rv:38.0) Gecko/20100101 `**`Firefox/38.0`**
Microsoft IE	`Mozilla/5.0 (compatible; `**`MSIE 10.0; Windows NT 6.2;`**` WOW64; Trident/6.0)`
Google Chrome	`Mozilla/5.0 (`**`Windows NT 6.2;`**` WOW64) AppleWebKit/537.36 (KHTML, like Gecko) `**`Chrome/43.0.2357.81`**` Safari/537.36`

Table 4 - Example of 3 different User-Agent field values

The four fields referenced (Accept, Accept-Encoding, Accept-Language and User-Agent) were the ones who presented the most differences between browsers.

HTTP header fields by themselves will not be enough to extract a reliable (and unique) fingerprint from each device as the universe of different combinations is not very wide. This technique will have to be complemented with others to be effective.

Advantages

The biggest advantage of using the HTTP headers is that every incoming request contains a set of them, making it easier for the webserver to collect the information. Therefore, no scripting is needed. This reduces the risk of a *fingerprinting* script being blocked, breaking the process of data retrieval.

It's completely silent, as the client's browser gives the information away.

Disadvantages

The HTTP fields cannot be used alone to create a fingerprint because there is no sufficient diversity that might ensure a unique signature. This technique should always be used complementary to others.

Moreover, the HTTP header fields are not mandatory in order to allow the webpage access. A user with a customized browser can clean all header files and have no information sent to the server on the Accept-Encoding, Accept and User-Agent fields. Such procedure could slightly affect the user experience but would not prevent the client to access the webpage.

3.3 Browser properties

System settings such as, the type of contents supported by the browser, the local clock time or the client's system default language can be externalized by the browser software. The frequent reason behind this giveaway of information is to provide a better user experience to the website client. In short, if the browser can negotiate with the website the best settings for the display of page contents, less work will have to be done by the human user.

The following ways of information gathering differ in their methods. Some rely on information voluntary sent by the browser and others ask the questions themselves.

3.3.1 Plugin and Mime-Type enumeration

In the JavaScript logic, the object `navigator` represents the browser and allows accessing its settings. When a website needs to query if a certain plugin exists in the client's system, in order

to properly display/run some type of content, it can retrieve an array of plugins by accessing to `navigator.plugins`.

Since each plugin can contain a series of mime-types that it supports, it is also possible to access the array of supported mime-types with `navigator.plugins[i].type`[28] (where "i" corresponds to the plugin currently being accessed). It is also possible to check if the plugin is active or not.

As stated above, not all browsers allow this type of queries. Of the various browsers tested, only *Mozilla Firefox* allows this kind of retrieval. The **Appendix I** (*Example of a browser plugin and mime-type list, retrieved with Javascript*) shows the result[29] of this kind of query, done by the *fingerprinting* awareness website *Panopticlick.*

As it is possible to see, this is a very extensive list, making this a good start for the building of a fingerprint of a client device.

The works of Eckersley (2010), Acar, et al. (2013), Nikiforakis, et al. (2014) and Mayer (2009) described this technique.

This technique retrieves system data and, simultaneously, produces different type of output depending on the browser being used. It can be used both for *browser fingerprinting* or *cross-browser fingerprinting*.

Examples:

The **Appendix I** (*Example of a browser plugins and mime-types list*) shows the output of a plugins and mime-types list collection, retrieved by the author from the website *Panopticlick*, available at https://panopticlick.eff.org.

Advantages

The list of browser plugins and mime-types is a good starting point for the *fingerprinting* process. It provides an extensive list of browser related information, including plugin versions, which adds diversity to the profiling process.

[28] While the property `type` returns the mime-type, the property `description` could (but not always) contain a description of the mime-type while `suffixes` presents the file extensions associated.
[29] The output was slightly edited for reader's convenience.

Disadvantages

The browser plugins are pieces of software that can be removed or updated (to new versions). The creation of a signature should not rely on a simple hash of the plugin list, or else the fingerprint would be rendered useless right after the first plugin update, or when a plugin is removed.

Browsers react differently to the `navigator.plugins` request. While browsers such as *Mozilla Firefox*, *Opera* and *QtWeb* provide a full list of plugins and mime-types, others like Microsoft IE are developed to only verify the existence of a specific plugin upon occurrence of a content that requires it, rendering the listing useless[30].

3.3.2 Do-Not-Track

The Do Not Track header was a proposed HTTP header field (DNT) with the objective of increasing the privacy of the webpage users.

The header contains a flag that indicates whether the client is willing to be tracked across websites. Unfortunately, the browser user has no control over whether the request is honoured or not, so the effectiveness of this measure is questionable.

The Do Not Track header can be retrieved using the JavaScript language, calling the value `navigator.doNotTrack`.

In the tests that were made, browsers returned answers with slight differences. These results would surely not be sufficient to create a fingerprint by themselves but they could help in the browser identification process.

Mayer & Mitchell (2012), Acar, et al. (2013) and Nikiforakis, et al. (2013) mentioned the Do-Not-Track header and agreed that its usefulness is questionable at best.

Because it produces slightly different answers across browsers (see examples below), this technique can add diversity to the *browser fingerprinting*.

Examples

The following example shows how different browsers implement differently the answer for what appears to be the same internal value (i.e. DNT not set).

[30] For Microsoft IE, the output for the `navigator.plugins` call is "No plug-ins are installed."

Browser	Do-Not-Track value
Mozilla Firefox	`Do-Not-Track: unspecified`
Microsoft IE	`Do-Not-Track: undefined`
Google Chrome	`Do-Not-Track: null`

Table 5 - Example of 3 different Do-Not-Track field values

Advantages

The kind of answer to the DNT request could help determining the browser brand, as long as the *fingerprinter* has a list of browsers with the DNT known answers.

Disadvantages

Aside from providing the flag to the DNT, this property does not offer much information. It can only be used together with other retrieved data.

3.4 Browser behaviour

Aside from the browser properties, that can be browsed by the website or are joyfully sent by the browser itself, some other information is possible to retrieve from the browser behaviour.

Browser vendors implement their software to be compliant with protocol specifications (such as HTTP, for example) and web standards, following the recommendations of bodies such as W3C, IETF or ISO. However, vendors are free to include their own logic in the code that is embodied in the browser software. This means that, the outcome of a certain operation in two different browsers might be the same, or so alike that the human brain can't tell them apart when, in fact, two different approaches were made.

The way that browsers handle mathematical operations or render graphics, for example, can allow differentiating them.

Advantages and disadvantages of the following techniques will be presented at the end, as both approaches share the same features.

3.4.1 Browser rounding and fractional pixels

The way that different browser handle math calculations, and rounding in particular, can be used to identify the client's browser and contribute to the *fingerprinting* process.

Kilgour (2014) showed that different browsers deal with percentage widths with decimal places and pixel values with decimal values in different ways.

The webpage where the study is presented (*Browser Rounding and Fractional Pixels*) contains a test page where the users can test their browser's behaviour. In this page, a series of calculations with percentage with decimal values and decimal pixel values are used. Those values are used as width and height parameters for graphical objects and finally the graphical objects are evaluated in order to understand the browser engine behaviour.

Being a browser benchmarking technique, this approach can contribute to perform *browser fingerprinting.*

Examples:

(Values taken from the website *Browser Rounding and Fractional Pixels*[31])

In the example shown in Table 6, the site calculates percentages of decimal values. It also creates multiple graphic boxes with fractional pixels. Finally, the site proceeds to measure the length of the objects actually rendered by the browser.

Browser	**Box 1 width percentage as calculated by the browser**
Mozilla Firefox	50.529%
Microsoft IE	50.52%
Google Chrome	50.5290112%
Browser	**Box 1 width in pixels as calculated by the browser**
Mozilla Firefox	669.5px
Microsoft IE	670px
Google Chrome	669.5px
Browser	**Box 2 width percentage as calculated by the browser**
Mozilla Firefox	50.329%
Microsoft IE	50.32%
Google Chrome	50.3290112%
Browser	**Box 2 width in pixels as calculated by the browser**
Mozilla Firefox	666.8499755859375px
Microsoft IE	667px
Google Chrome	666.859375px

Table 6 - Example 3 different results for browser rounding and fractional pixels

[31] Available at http://cruft.io

Advantages

Unlike other browser behaviour-related approaches that are affected by the system usage (e.g. CPU load or allocated RAM memory) at a given time (like the measuring of the browser's performance), this technique is unaffected by such constraints.

Disadvantages

This approach is not enough to create a unique signature. At most it can help identifying the browser brand and model. Requires a comparison database, to relate the measured patterns with the browser brand.

3.4.2 Canvas *fingerprinting*

The canvas *fingerprinting* technique applies a principle similar to the previously shown: the rendering of a graphical object by different systems (browsers) produces different output, and therefore different fingerprints.

In the work of Mowery & Shacham (2012), the authors showed that the rendering of fonts and graphical elements had slight variations between different browsers, what allowed the extraction of an individual signature.

This technique takes advantage of the element `<canvas>`, an element of HTML5 that provides an area on the screen which can be drawn upon programmatically. Two types of content are proposed to be rendered in the `<canvas>` element: text and shapes. For the text content a Cascading Style Sheet Version 3 (CSS3) can be used in order to allow the browser to download the required font, in case it is not installed on the system. The drawing of polygons uses the WebGL API. After rendering the objects, the resulting images are captured and analysed.

The image comparison that Mowery and Shacham applied were pixel-level difference and difference maps[32]. From the 294 experiments on Amazon's Mechanical Turk, 116 unique fingerprint values were observed. The authors justified that the modest ratio of identifications (around 39%) was caused by the little variation in browser and OS in the device population involved in the experiment.

[32] Techniques used to compare two images. Consist in subtracting the colour values of a certain pixel in one image by the correspondent in the other image. The resulting image map will determine how alike the two images were.

In a study about canvas *fingerprinting* and evercookie feasibility (Acar, Eubank, Englehardt, Juarez, Narayanan & Diaz, 2014), the authors crawled the Top Alexa[33] 100,000 sites and found that "more than 5.5% of crawled sites actively ran canvas *fingerprinting* scripts on their home pages."

This technique can contribute to perform *browser fingerprinting*.

Examples

Few implementations can be found online of HTML5 canvas *fingerprinting*. The website *BrowserLeaks.com*[34] provides one, but warns that the comparison database, to which the incoming signatures will be compared to, is not complete enough to provide a complete coverage of the whole universe of browsers and does not collect new signatures.

Table 7 shows the browser signatures extracted from the testing browsers.

Browser	Signature	General conclusion (of the site)
Mozilla Firefox	5525E5D4	"It is very likely that you are using [**Firefox**] on [**Windows**]"
Microsoft IE	62939B59	"It is very likely that you are using [***Internet Explorer***] on [**Windows**]"
Google Chrome	F921F32B	"Your system fingerprint appears to be unique (…)"[35]
Opera	C0042A00	"It is very likely that you are using [***Opera***] on [**Windows**]"
QtWeb	FCCD5D4B	"It is very likely that you are using [**Chrome**] on [**Windows**]"
Midori	04D7B814	"Your system fingerprint appears to be unique (…)"
Chromium	F921F32B	"Your system fingerprint appears to be unique (…)"

Table 7 - Example of 7 different browser signatures for canvas fingerprinting

By the analysis of these examples it is possible to see how heavily this technique depends on a wide and constantly updated database of signatures. While some of the most popular browsers

[33] Refers to "Alexa Internet, Inc.", a subsidiary company of Amazon.com, which provides commercial web traffic data.

[34] Available at https://www.browserleaks.com/

[35] The whole text reads "Your system fingerprint appears to be unique, yet we don't collect signatures here, just check."

were correctly detected (e.g. *Mozilla Firefox*, *Internet Explorer* and *Opera*), other popular browsers like *Google Chrome* were not successfully identified.

Chromium and *midori* were not identified as well and *QtWeb* was even mistaken with *Google Chrome*, a browser with whom it shares no common platform!

Advantages

This is another browser behaviour technique that is not affected by the computing workload at the moment of the data collection.

Disadvantages

The technique is only reliable when backed up by a database that maps the whole browser signature universe and that is constantly collecting new signatures in order to be updated. This requires a considerable effort from the *fingerprinter* given that one must be aware of new coming browsers and of changes to the behaviour of the existing ones.

Having that said, any change made by the vendor to the internal mechanisms of the browser's computing architecture might have a dramatic effect on the behaviour of the software. Once again, without a constant update of the browser trends, the *fingerprinter* might not be able to keep up with the incoming patterns.

3.4.3 Browser Performances

Mowery's 2011 work (Mowery, et al. 2011) described a technique that measured the timing differences of multiple operations in the core of the JavaScript language. According to the authors, it would be possible to distinguish not only browser versions but also micro architectural features not normally exposed to JavaScript.

The objective was to be able to identify browser family and version, operating system and architecture type. The technique shown was based on a benchmarking approach. Browsers had to execute 39 individual JavaScript tests whose completion time was measured in milliseconds and added to a 39-dimensional vector which characterized the performance of the tested machine. Benchmark suites, such as *V8* and *SunSpider*, were created to measure scripting speed. In order not to be influenced by external processing (that would increase the measured time), each test was run five times and the minimum positive time for each test. Differences in processor architecture and clock speed impacted these timings significantly, therefore, allowing the creation of a signature.

From a universe of 1015 user configurations, the authors were able to identity 79.8% of the family and browser version. Since this technique only allowed operating system detection within a particular browser version, the authors decided to go with Firefox 3.6, (as it was reliably detectable in the test). From this subset, the authors noted that Windows operating systems showed the biggest identification rate (98.5%). The identification of the processor architecture had also to be done towards a specific browser version and showed a success rate of 45.3%.

This technique can contribute to perform *browser fingerprinting*.

Examples:

There are currently no webpages available implementing this specific technique (i.e. using the 39 tests). However, benchmarking suites such as *V8* and *SunSpider* are available in webpages where the clients can test their browsers' performances.

The current author performed 20 tests in the website *V8 Benchmark Suite*[36] and 10 tests performed in *SunSpider JavaScript Benchmark*[37] with a single browser (*Mozilla Firefox*), to verify the practicability of Mowery's technique. A part of the results can be seen in the **Appendix II** (Example of JavaScript Performance Benchmarkings) that, due to size limits couldn't be shown entirely in this section (12 tests of *V8* and 2 of *SunSpider* are shown).

The results did not show much consistency across the tests, when using the same configuration set (i.e. the combination of browser, operating system and CPU architecture). This might mean that the test sample was not wide enough for the average of the results obtained to tend to a certain value (according to the law of large numbers), therefore, more tests would have to be performed (Mowery mentioned 39 tests) that would allow to find an average that would later be compared with an existing database.

Advantages

An advantage about his approach is that it does not rely on information that is communicated by the browser. It rather observes the behaviour and registers the results. This makes the job harder for those trying to develop anti-*fingerprinting* browsers, because the simple mitigation of the data sent is not enough to protect against this technique.

[36] Available at http://v8.googlecode.com/svn/data/benchmarks/v7/run.html
[37] Available at https://www.webkit.org/perf/sunspider-1.0.2/sunspider-1.0.2/driver.html

Disadvantages

The performance information collected from the user's system is highly dependable on the processing being done at that moment and, therefore, different tests might show large discrepancies in the time values.

On other hand, the authors of this work agree that *[o]ne of the largest weaknesses in [the] approach is that the fingerprinting time is very large - usually over 3 minutes.* While the tests are being performed the whole client system is affected by a sudden degradation of performance, mostly noticed by the browser's lack of responsivity. This undermines any possibility of using this technique on a stealthy way.

3.5 Operating system features

Certain browser plugins have the ability to access operating system related information. Java and Flash plugins are known to access system settings in a fashion that is not totally privacy-friendly, as it bypasses the browser's controls. As referred in Chapter 3.2, the TorProject team (developers of the Tor Browser) warns their users about the threats to anonymity associated with the use of Java and Flash plugin.

The collection of operating system related information using these technologies is mentioned by Eckersley (Eckersley, 2010) and Nikiforakis (Nikiforakis, et al. 2013), although not extensively debated.

The universe of information possible to retrieve from the system with the Java technology is significantly greater than the one given by Flash.

3.5.1 Using Flash plugin

The online *ActionScript 3.0 Reference for Adobe Flash Platform* website[38] informs that:

> *The Capabilities class provides properties that describe the system and runtime that are hosting the application.* (…) *By using the Capabilities class to determine what capabilities the client has, you can provide appropriate content to as many*

[38] Available in http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/index.html

users as possible. When you know the device's capabilities, you can tell the server to send the appropriate SWF files or tell the SWF file to alter its presentation.

The Capabilities class allows the browser to retrieve information about the Flash plugin itself, the system that hosts it, and a set of system and browser features.

The data collection through the Flash plugin allows *browser fingerprinting* and *cross-browser fingerprinting*, because there is plugin-related information (browser configuration) and system information being retrieved.

Examples

The current author retrieved a list of browser and system features, collected by the website BrowserLeaks.com, for the *Mozilla Firefox* and *Google Chrome* browsers. **Appendix III** (Example of OS and browser features, retrieved with Flash Plugin) shows the subtle differences that can be found between the information captured for each one, namely, the information about the Flash plugin itself.

Note: It is also possible to collect the list of system fonts with the use of the Flash plugin. However, that will be presented in the Section 3.7 (Font detection/enumeration).

3.5.2 Using Java

The Java plugin can provide information about the local Java Virtual Machine (JVM).

This technique provides system-related information, therefore allowing to perform *cross-browser fingerprinting*.

Examples

The author collected a list of system features, captured by the website *BrowserLeaks.com*, for the browsers *Internet Explorer* and *midori*. **Appendix IV** (*Example of OS and Java features, retrieved with Java Plugin*) shows only minor differences that can be found between the information captured for each one, namely, the information about the JVM Uptime and JVM Start Time.

Advantages

This technique provides a fair amount of system-related information that is collected by the plugins, via API calls to the operating system. Because the calls are directed through the plugins

and as long as the plugin is active, it's difficult to implement mechanism on the browsers that could prevent this information collection – thus, making this a reliable tool for *fingerprinter*s.

An interesting difference between both approaches is that the Java plugin returns the information it reads from the JVM which, by default, is common to all Java-dependable software on a particular system, on the other hand Flash collects information directly from the Flash plugin. This allows Flash to retrieve browser-signatures that will be different if different plugins are being used (see **Appendix III**).

Disadvantages

The approach using Java is the one that provides the most verbose information (i.e. actual system data, instead of Flash's Boolean properties), making it more suitable for creating a system signature. However, because most of the browsers are currently prompting users about whether they are sure to run Java Applets, it's difficult to collect this information in a stealthy fashion.

3.5.3 Clock skew

In Kohno, et al. (2005) the authors suggested the *fingerprinting* of devices through comparison of the clock skew between the client system and the server clock. According to the authors, it would be possible to exploit *small, microscopic deviations in device hardware: clock skews*, even when the *measurer is thousands of miles, multiple hops, and tens of milliseconds away from the fingerprinted device, and when the fingerprinted device is connected to the Internet from different locations.* The authors stressed that *one can use [the] TCP timestamps-based method even when the fingerprintee's system time is maintained via NTP.*

The technique was based on the commonly used TCP timestamps option (from the TCP stack), which means that every request from the client would have the local time attached to it. Several tests were made with clients in different countries to understand the effects of topology on skew estimates. The results showed that the *clock skew estimation techniques are independent of access technology and topology.*

As clients can have multiple, possibly independent, clock, the technique also measured the difference between the client's system time and the internal clock to the device's TCP stack (TCP timestamp), which is a virtual clock not necessarily synchronized with the system clock.

This technique would allow to perform *cross-browser fingerprinting*.

Examples

No examples were retrieved for this technique.

Advantages

This technique allows the identification of different devices behind a NAT and even the detection of virtual *honeynets* (the performed test showed that virtual machines did not have constant, or near constant, clock skews).

Disadvantages

The measuring duration of the tests showed in the work of Kohno, et al. (2005) range up to 120 minutes which is a rather high expectancy time for a use to be browsing on a certain domain.
Besides that, the authors show the limitations of this technique in the following observation:

> *With respect to tracking individual devices, we stress that our techniques do not provide unique serial numbers for devices, but that our skew estimates do provide valuable bits of information that, when combined with other sources of information such as operating system fingerprinting results, can help track individual devices on the Internet.*

3.6 Hardware features

Being application software, browsers require the support of operating systems in order to be able to function and interact with the hardware level. By using APIs, the browsers are able to interact with the operating system and, through it, interact as well with the hardware components.

Therefore, browsers can read hardware settings, such as the screen resolution, number of CPU cores, the amount of dedicated memory or the identification of network interfaces.

Although the combination of the above settings is far from being able to provide a unique set, once combined with browser settings and other *fingerprinting* methods they can contribute to the creation of a unique signature

3.6.1 Screen Properties

The retrieval of the screen properties is possible to achieve through JavaScript or Flash.

With JavaScript, the object `window.screen` allows the retrieval basic information about the screen of the client. Unlike most of the other web-based methods to get device information, this one actually manages to collect information of hardware settings.

The `window.screen` object allows getting information about screen height (in pixels), width, and bit depth of the colour palette available for displaying images.

Acar, et al. (2013) addressed this type of data collection with a *fingerprinting* detector, *FPDetective*.

Examples

Table 8 shows examples of 3 different browser behaviours when requested to output the screen properties.

Mozilla Firefox	midori	QtWeb
availHeight: 860	availHeight: **900**	availHeight: 860
availWidth: 1600	availWidth: 1600	availWidth: 1600
height: 900	height: 900	height: 900
width: 1600	width: 1600	width: 1600
colorDepth: **24**	colorDepth: 24	colorDepth: **32**
pixelDepth: **24**	pixelDepth: 24	pixelDepth: **32**

Table 8 - Example of 3 different screen property values for the same system

Note: Surprisingly, the test returned slight variations in the results of some browsers. The variation in the `availHeight` value has to do with the different way that browsers evaluate the available height: most consider the available height as the difference between the total height and the height of the task bar. Since the system used to perform the tests ran on a Windows 8 machine, with a task bar at the bottom, all tested browsers considered the available height as having 40 pixels less, except the *midori* browser.

Flash allows the retrieval of screen properties as well. The type of information includes screen resolution, screen colour, screen DPI and pixel aspect.

Examples

Table 9 shows the screen values collected with the Flash plugin for 3 different browsers.

Properties	Mozilla Firefox	midori	QtWeb
Screen Resolution	1600x900	1600x900	1600x900
Screen Color	color	color	color
Screen DPI	72	72	72
Pixel Aspect Ratio	1.0	1.0	1.0

Table 9 - Example of 3 different values for Flash screen properties

The browsers returned the same results, what indicates that the plugin might have consulted this data from the operating system.

This technique is based in the system's properties, therefore contributing to *cross-browser fingerprinting*.

Advantages

The collection of screen size values can allow the *fingerprinter* to infer if the client's device is a smartphone, a tablet or a two-screen desktop computer (the widths of both screens would be added).

Disadvantages

As was seen in the first example (JavaScript approach), caution must be taken when processing these values. Different browser implementations can change the measuring rules, therefore producing different signatures and jeopardizing any cross-*browser fingerprinting* attempt.

3.6.2 CPU and RAM memory

The Java technology allows to retrieve information about the number of available processors, and the amount of free, max and total RAM memory.

Browsers showed different behaviour for this data retrieval: while *Internet Explorer*, *Mozilla Firefox* and *midori* provided the requested information, others returned no data at all.

This technique contributes to *cross-browser fingerprinting*, as Java accesses OS interfaces.

Example

The examples in Table 10 show the different results for 3 distinct browsers.

CPU and RAM	Internet Explorer	Mozilla Firefox	midori
Available Processors	2	2	2
Free Memory	**8.135.728**	**10.363.232**	**11.060.632**
Max Memory	259.522.560	259.522.560	259.522.560
Total Memory	16.252.928	**16.384.000**	16.252.928

Table 10 - Examples of 3 different results for CPU and RAM

The different value for “Free Memory” is somewhat expected, as the system manages memory continually, resulting in different values. The difference in the “Total Memory” between *Mozilla Firefox* and the other two is less clear. Only the properties “Available processors” and “Total Memory” seem useful for creation of a signature.

Advantages

This technique provides very specific system-related information that might help characterize the type of web user (e.g. a tuned up system might indicate a gamer or a graphics designer), which might be interesting information for website owners or advertisers.

Disadvantages

Properties like “Free memory” or “Total memory” should not be relied upon.

3.6.3 Network Interfaces Enumeration

Java enables the retrieval of the name and status of physical and virtual network.

Among the type of information that is returned is the name of the interface, the status (i.e. if the interface is enabled or not), the Maximum transmission Unit (MTU), if is point-to-point, if supports multicast, if is loopback and if the interface is virtual.

The information retrieved by this technique is system-related, therefore it’s suitable to *cross-browser fingerprinting*.

Examples

A test performed by the current author with the *Internet Explorer* browser in the website BrowserLeaks.com produced a list of 106 (!) network interfaces for a single system. Due to space constraints, the full list of interfaces is located at the **Appendix IV** (*Example of Network Interface Enumeration with Flash Plugin*).

Note: The reason for the high number of network interfaces has to do with the interfaces the operating system (Windows 8, for the current case) defines for internal use.

Advantages

This technique has the advantage of being able to quickly identify if the device being used at the current time is the same used before. This feature can have practical application in situations where a website wants to make sure that the user that signs in using a different hardware from the usual is, in fact, the real user. In order to be able to detect the new hardware, the website must have a set of signature of former hardware signatures associated to that user. In case of an incoming new signature, the website can prompt the user with a security question or request a multi-channel authentication.

Disadvantages

The hardware's collectable information is slim and there is not enough diversity in order to build a unique signature. Therefore, *fingerprinting* by collection of hardware features must rely on additional techniques in order to be effective.

3.7 Font detection/enumeration

Different operating systems have their own font sets and, on top of that, software suites such as Microsoft Office or Adobe Creative Suite usually add their own set of fonts to the system. Additionally, users can add their own fonts to the system as well. The resulting set of fonts can be diverse enough to build a signature or, at least, allow the identification of the operating system. The work of Boda, Földes, Gulyás and Imre (2012) showed how the universe of fonts is affected by the installation of Office and Adobe suites.

Fingerprinting by font enumeration is mentioned in the works of Acar, et al. (2013), Eckersley (2010) and Nikiforakis, et al. (2013), making this one of the most commonly mentioned *fingerprinting* techniques, together with plugin detection.

Fonts can be enumerated using JavaScript, Flash, Silverlight or Java, with different levels of efficiency.

Because system fonts are managed at the operating system level, this technique allows to perform *cross-browser fingerprinting*.

As stated in the website *latit.lab*, the font detection done through JavaScript and CSS takes advantage of the fact that *each character appears differently in different fonts. So different fonts will take different width and height for the same string of characters of same font-size*. The fingerprinted tries to generate a string of text for each font he has in his comparison database. If the font does not exist in the system, the fall back font will be rendered instead. By measuring

the output the *fingerprinter* will be able to identify if the rendered font is different from the fall back and, if that's the case, existent in the system.

Usually, the Flash font enumeration uses a combination of ActionScript and JavaScript calls. The website *hasseg.org*[39]describes a possible way to achieve this.

Silverlight uses the object collection `Typeface`[40] to retrieve the system's font set. Java can also retrieve the set of fonts by using the `getAvailableFontFamilyNames()` method, of the `GraphicsEnvironment` object[41].

In a series of tests performed on the website *BrowserLeaks.com*, the results for font enumeration were the following, for each of the previously mentioned technologies:

Browser	JavaScript + CSS	Flash	Silverlight	Java
Internet Explorer	276 of 512 fonts	276 fonts	467 fonts	265 fonts
Mozilla Firefox	232 of 512 fonts	No result	467 fonts	265 fonts
Google Chrome	231 of 512 fonts	**270 Fonts**	Silverlight does not exist	Java Plugin does not exist
Opera	264 of 512 fonts	276 Fonts	467 Font	No result
Chromium	231 of 512 fonts	**270 Fonts**	No result	No result
midori	247 of 512 fonts	276 fonts	467 fonts	265 fonts
QtWeb	284 of 512 fonts	276 fonts	No result	No result

Table 11 - Example of 7 different sets of fonts detected

Note: The fields that state "No Result" refer to tests where the browser did not return a result.

A couple of remarks can be made here. In first place, it's interesting to see that different browsers showed different results when using the same technologies (e.g. JavaScript+CSS and Flash). The duo JavaScript + CSS showed a worrying variance between the different browsers – 6 different results sets out of 7 browsers. This appears to occur because the JavaScript interpreters built in each one work differently. For instance, while *Internet Explorer* detected 11 *Arial*-related fonts (e.g. *Arial*, *Arial Baltic*, *Arial Black*, etc.), *Mozilla Firefox* only identified 8 fonts and Chrome merely 4 fonts. This behaviour was also detected with other font-families

[39] Available at http://hasseg.org/blog/post/526/getting-a-list-of-installed-fonts-with-flash-and-javascript/
[40] See more at https://msdn.microsoft.com/en-us/library/system.windows.media.fonts.systemtypefaces%28v=vs.95%29.aspx
[41] Example of Java retrieval of all the available fonts available at http://alvinalexander.com/blog/post/jfc-swing/swing-faq-list-fonts-current-platform

such as *Bodoni*, *Franklin Gothic* or *Gill Sans*. It seems that some interpreters consider certain variations as being the original font, decreasing the total number of identified fonts.

Java and Silverlight remained coherent between the browsers that supported them.

Advantages

The enumeration of fonts is probably the data collection technique that provides the biggest amount of data from the client's system. Although the font sets feature information that can change (e.g. fonts might be added because new software was installed), the *fingerprinter* can still find this risk negligible and decide to create a signature from the list of fonts.

*Opera*ting systems can also be inferred from the font sets, as long as the *fingerprinter* has a database of specific operating system featuring fonts. The same applies for browsers.

Disadvantages

In order to perform the *fingerprinting* using JavaScript, a database of fonts is needed for comparison. A complete identification of the fonts depends on having the most extensive database of fonts possible, to encompass all fonts that possibly might appear. In fact, the less common fonts are the ones who are able to provide the diversity necessary for a signature.

3.8 Cached Objects

Whenever browser cached objects are mentioned the first thought that comes to mind is: HTTP cookies (probably simply, cookies). There is a good reason for this. Cookies have been widely used since the nineties for storing data on the client side, whether for storing legitimate session data or for the, not so legitimate, third-party storing of data.

But even with all the privacy concerns inherent to cookies, they still have a good thing on their side: browsers and users understand the concept of cookies and, thanks to this, preventive and reactive measure can be taken against the misuse of cookies.

However, the evolution of the web languages and the integration of plugins within the browsers brought new ways of storing (and reading) data in the client side, alternative to the classic cookies. Originally, all of these new data storing mechanisms were aimed to make the user experience more pleasant, by remembering preferences and settings. Of course, it was just a matter of time before these mechanisms would be exploited.

The technique of using stored objects on the client side as a way to uniquely identify a device is fairly similar across the technologies that have this ability: first the *fingerprinter* checks the

cached object holder for the presence of any previous unique ID (stored from a former visit), if no ID exists the website uses the cached object to store a somewhat random but unique ID that will allow further visits to be tracked and linked to this device.

Flash and HTML5 have their own mechanisms for storing data. These will be described furthermore.

Strictly speaking, the exploitation of cached content cannot be considered a *fingerprinting* technique. After all, devices are not being identified by a set of unique features but rather, by a distinctive mark that has been previously etched in each one. But let's not forget that, for the unique identifier to be stored in the system in the first place, it was necessary to perform a previous verification of "non-existence" of that device in the collected universe, which is by itself a *fingerprinting*-like processing.

Furthermore, the ultimate goal of exploiting cached content is the same of *web-based fingerprinting*: the tracking of devices in a passive way.

3.8.1 HTTP cache

Before explaining the Flash and HTML5 mechanisms a mention should be done to the web cache technology and the exploitation that can be done of it.

The caching of webpage contents allows to reduce bandwidth usage, waiting times and server load. When requesting a resource it has seen and stored before, the client indicates the version of the stored object and asks the server if it should use or download a newer. The server either will answer with a 304 status code (Not Modified), meaning that the resource has not been modified since the version specified by the request headers and that there is no need to retransmit the resource, since the client still has a previously-downloaded copy, or, with a 200 status code (OK) which is the standard answer with all the contents of the webpage.

In Zalewski (2012), an exploitation for this behaviour was described.

If the server can indicate the client browser that it should or not replace a file stored on its system, then it has a certain degree of control over that file and can use it for *fingerprinting* purposes.

In a practical example, the author showed how the *fingerprinter* can dynamically generate a JavaScript dependency file containing a unique ID for each client requesting the file for the first time. The website code will have a script to send a HTTP Request using the unique ID stored in the JavaScript cached file. Whenever the browser asks the server if a new version of the JavaScript dependency exists, the server will answer negatively, perpetuating the sending of requests with the ID.

One way to counter this technique is to clear the cached content and retrieve a whole new version of the page (and having a new unique ID being set!).

Because each browser stores its HTTP cache is in a specific folder, this means that stored objects only allow the identification of the browser that stored them (*browser fingerprinting*).

Examples

No examples were retrieved for this technique.

Advantages

Most browsers allow the storing of objects by default, especially when used in mobile devices, making this technique reliable. As was stated before, systems do this to reduce bandwidth usage and to increase access speed as well.

Disadvantages

Devices configured not to store cache or to delete cached objects when the browser session is over will not be affected by this technique (although those that allow the storing during session could be tracked throughout that period).

3.8.2 Browsing history

Olejnik, Castelluccia and Janc (2012) proposed a *fingerprinting* technique based on the profiling of the user's browsing history.

The collection of the URLs in the browser's history is done by using the technique described by Janc & Olejnik (2010), which allows:

> *(...) an attacker to determine if a particular URL has been visited by a client's browser through applying CSS styles distinguishing between visited and unvisited links. (...) the attacker must supply the client with a list of URLs to check and infer which links exist in the client's history by examining the computed CSS values on the client-side.*

Because creating a signature of the browsing history would be rather useless, the authors decided to create "history profiles" (that act as fingerprints) where URLs were categorized by website subject (e.g. Shopping, Entertainment, Travel, Vehicles, etc.). In the observations of the researchers:

> *(…) a large number of users have unique personal browsing interests even when analysed using the more coarse-grained category metric (…)In a real scenario of an advertising provider, multiple repetitions of each category in the profiles are likely used to enumerate the strength of interest in the category which provides additional information*

This test was performed in a universe of 368.284 web histories, and more than 69% of users had a unique fingerprint.

This technique can contribute to *browser fingerprinting*.

Examples

No examples were retrieved for this technique.

Advantages

This technique resembles the biometric *fingerprinting* techniques that aim to profile the human user behaviours, rather than browser or device properties. In that sense, this is a very appealing approach for advertisers that get more information with less processing on their side (i.e. if browsing profiles are already categorized into website subjects the advertiser won't have to associate signatures with hits).

Disadvantages

The authors "believe that Web browsing preferences can be used as an efficient behavioural fingerprint which is in many cases stable over time". The question is how long can it take to have a stable browsing fingerprint? The technique assumes that the user accesses the website frequently, creating temporary profiles for sporadic users that will not get to be useful.

On other hand, if a device is shared by multiple users using the same account, the technique might not be able to converge the patterns into a single profile, due to the different browsing behaviour of each user.

3.8.3 Local Shared Objects (Flash cookies)

Using previous versions of Flash, developers could save information between sessions by using 'normal' cookies, but the process was considered difficult for developers to implement - creating a cookie required the use of a language outside Flash (like JavaScript or ASP). In the Flash MX version, Macromedia introduced the Local Shared Object (LSO), which provided an easier way to store information (i.e. only required the use of ActionScript).

Local Shared Objects provide the only method by which a Flash application can store information on a user's computer. Intended uses of the object include storing a user's name, a favourite colour or the progress in a game.

This technique allows to perform *browser fingerprinting*.

The work of Nikiforakis, et al. (2013), Acar, Eubank, Englehardt, Juarez, Narayanan and Diaz (2014) and Mayer & Mitchell (2012) showed how LSO can be used to track users.

The actual information is stored in a .SOL file in a specific directory on the user's computer. Using the flash configuration tool, the users can decline Flash cookies but, unfortunately, few consumers are aware of where Flash cookies are stored or how to control their use and there is no mechanism to set an expiration date on Flash cookies.

The Electronic Privacy Information Centre (EPIC) warns for the risks of identification of individuals in an article regarding Local Shared Objects[42]. According to EPIC:

> *(...) the Flash movie can create a unique ID and store that ID in a Flash cookie on a user's computer. The Flash movie can then communicate this information to a database, or other applications. Subsequent visits by the same users could be tracked by reading the ID contained in the Flash cookie.*

And, in order to prevent the storing of Flash objects:

> *(...) users can adjust preferences on a per site basis in the Macromedia Website Privacy Settings Panel. Using this tool, Flash cookies can be completely disabled or allowed on a per domain basis.*

The Windows system also allows to manage preferences from the Control Panel.

Figure 3 depicts a screenshot of the Local Storage Settings by Site screen is shown below. Notice the three cookies already resident in the system.

[42] Article available at https://epic.org/privacy/cookies/flash.html

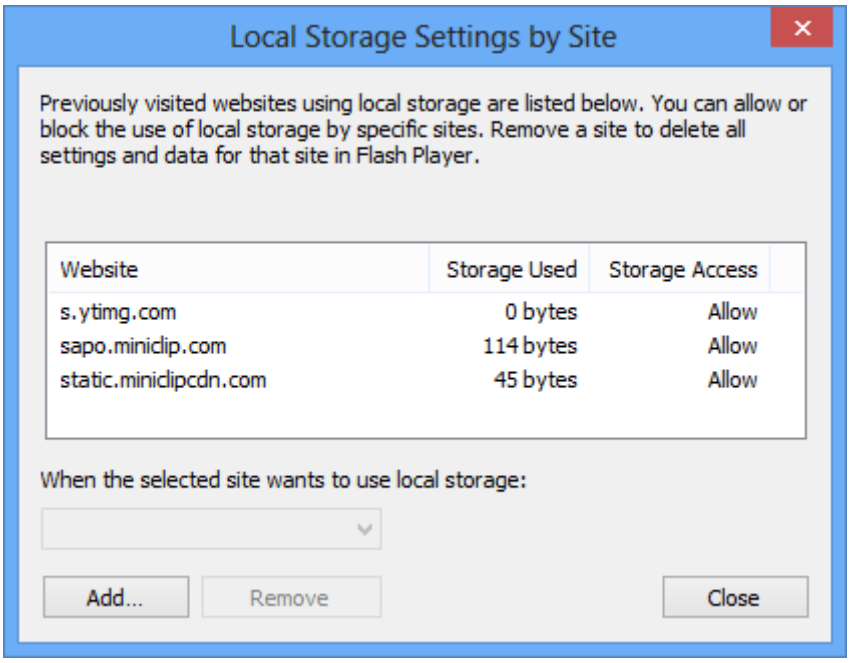

Figure 3- Screenshot of Local Storage Settings by Site (Flash).

Examples

No examples were retrieved for this technique.

Advantages

Flash cookies are a powerful way to track users because they are still not properly addressed by browsers in the sense that their management is not trivial (i.e. management is not done together with HTTP cookies). This lack of proper management paves the way for exploiting this functionality for tracking or *fingerprinting*.

Disadvantages

Because it requires the storing of information, this technique is considered intrusive. The use of this mechanism must abide to Article 5(3) of Directive 2002/58/EC, amended by the Directive 2009/136/EC (also known as the ePrivacy directive), which requires prior informed consent for storage or access to information stored on a user's terminal equipment.

3.8.4 Web Storage (HTML5 cookies)

HTML5 introduced[43] two related mechanisms, similar to HTTP session cookies, for storing name-value pairs on the client side: `sessionStorage` and `localStorage`. According to the *HTML Living Standard*[44]:

> *Storage object provides access to a list of key/value pairs, which are sometimes called items. Keys are strings. Any string (including the empty string) is a valid key. Values are similarly strings.*

The `sessionStorage` is designed for scenarios where the user is carrying out a single transaction, but could be carrying out multiple transactions in different windows at the same time. Since the data is only stored during session time, it has no useful application for *fingerprinting*.

On the other hand, the `localStorage` mechanism is designed for storage that spans multiple windows, and persists after the browser is closed. In particular, Web applications that wish to store megabytes of user data, such as entire user-authored documents or a user's mailbox, on the client side for performance reasons.

This mechanism allows the *browser fingerprinting*.

Websites like Youtube.com or Facebook.com use this type of storage. The works of Roesner, Kohno & Wetherall (2012) and Acar, et al. (2014) refer to the use of the `localStorage` mechanism.

The `localStorage` objects must be managed independently from the normal browser cache and only a few browsers have this feature built-in (e.g. *Opera* and Chrome).

An example of *Opera*'s management of `localStorage` objects is shown in the screenshot below.

[43] Originally part of the HTML 5 specification, Web Storage has its own separate specification nowadays, available at http://www.w3.org/TR/webstorage.
[44] Available at https://html.spec.whatwg.org/

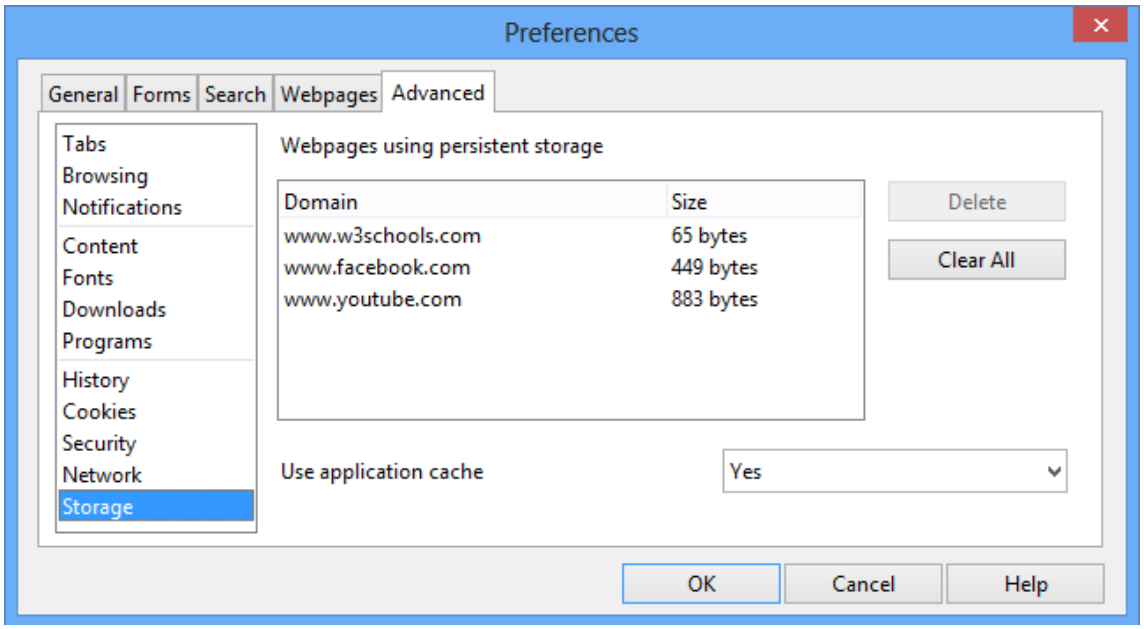

Figure 4- Screenshot of Opera's Web Storage management.

Examples

No examples were retrieved for this technique.

Advantages

HTML's `localStorage` might be a concept somewhat obscure to most web users. The functionalities of history, HTTP cookies and (normal) cached content cleaning, available in most browsers nowadays, might trick users into thinking that, once used, all browsing content related data will be successfully wiped from the system. Until browsers start alerting the users of this data storing and provide a simple mechanism to manage this type of data, users will be exposed to the possibility of having a persistent ID etched to their browser.

Disadvantages

In their nature, Flash's Local Shared Objects and HTML's `localStorage` are cookies. That means that the use of such mechanisms falls under the Article 5(3) of Directive 2002/58/EC, amended by the Directive 2009/136/EC, which requires prior informed consent for storage or access to information stored on a user's terminal equipment. In other words, websites using Flash or HTML5 cookies must ask users if they agree with the storing of data before the site starts to use them, risking penalties when not abiding to these obligations.

3.9 Taxonomy

The techniques previously stated show that the collection processing can focus on the browser features (*browser fingerprinting*) or in the underlying system, disregarding the browsing being used (cross-*browser fingerprinting*).

The following image shows the techniques that allow *browser fingerprinting*.

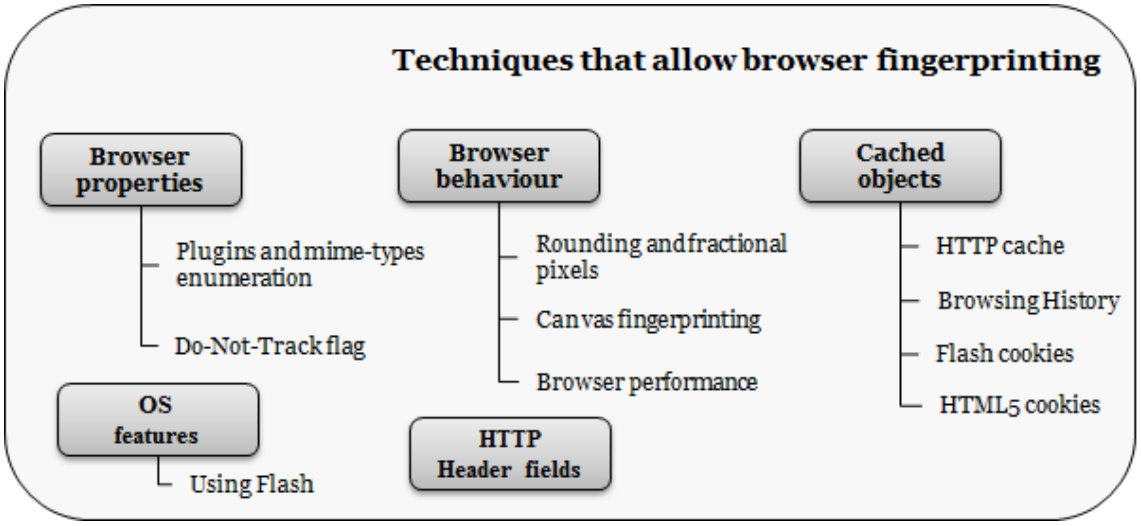

Figure 5 - Technologies that allow browser fingerprinting

Note: The Flash retrieval of OS features also comprises data about the Flash plugin itself, therefore, providing information about the browser.

The techniques contribute to cross-*browser fingerprinting* are shown in the image below.

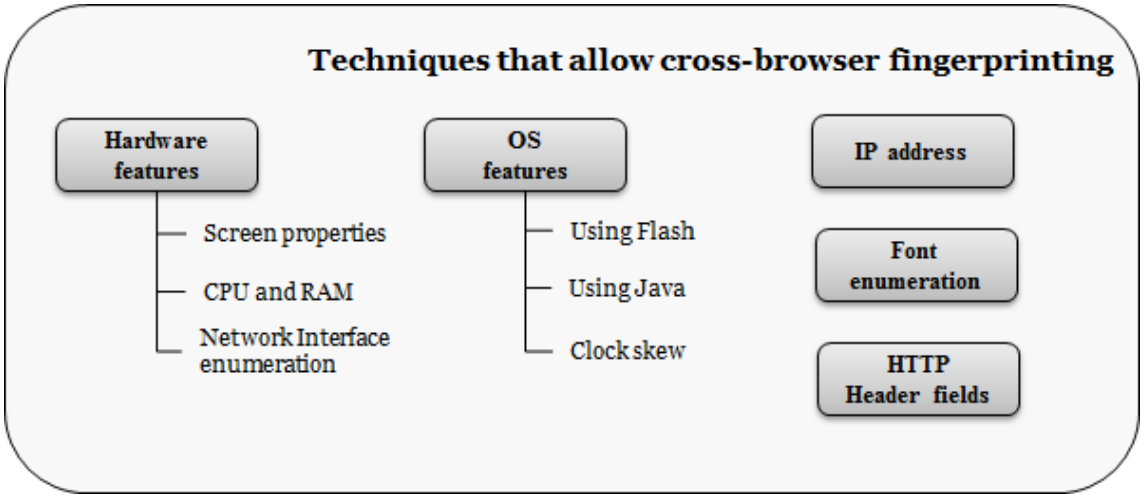

Figure 6 - Techniques that allow cross-browser fingerprinting

Note: Notice that HTTP Header fields are in both groups. This is because properties such as the "User-Agent" field provide both browser and system information.

The following image depicts a schematic with the distribution of techniques between passive and active approaches.

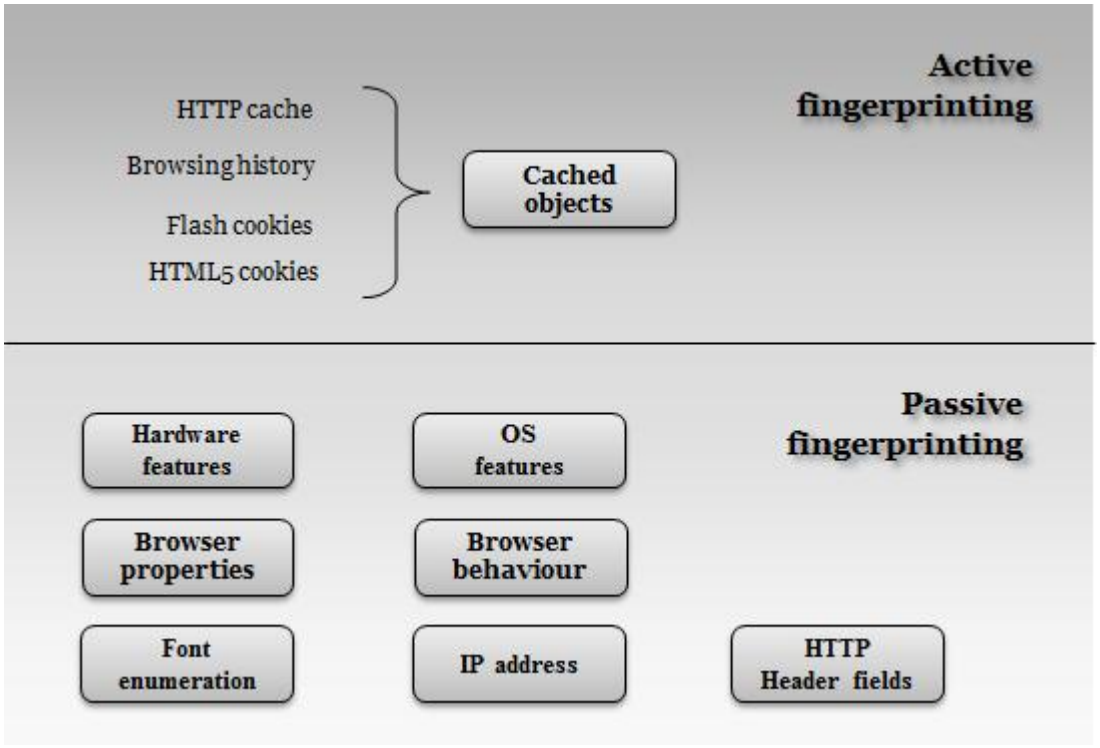

Figure 7 - Passive and active fingerprinting techniques

The following table depicts the taxonomy of the techniques studied.

Technique		*Browser fingerprinting*	*Cross-browser fingerprinting*	Active / Passive	Needs comparison database
IP address		NO	YES	PASSIVE	NO
HTTP header fields		YES	YES	PASSIVE	NO
Browser properties	**Plugin enum.**	YES	NO	PASSIVE	NO
	Do-Not-Track	YES	NO	PASSIVE	NO
Browser behaviour	**Rounding pixels**	YES	NO	PASSIVE	YES
	Canvas	YES	NO	PASSIVE	YES
	Performance	YES	NO	PASSIVE	YES
Operating system features	**Flash plugin**	YES	YES	PASSIVE	NO
	Java plugin	NO	YES	PASSIVE	NO
	Clock skew	NO	YES	PASSIVE	YES
Hardware features	**Screen properties**	NO	YES	PASSIVE	NO
	CPU and RAM	NO	YES	PASSIVE	NO
	Network interfaces	NO	YES	PASSIVE	NO
Font enumeration		NO	YES	PASSIVE	YES
Cached objects	**HTTP cache**	YES	NO	ACTIVE	NO
	History	YES	NO	ACTIVE	NO
	Flash cookie	YES	NO	ACTIVE	NO
	HTML5 cookie	YES	NO	ACTIVE	NO

Table 12 - Taxonomy of the web-based fingerprinting techniques

The taxonomy table evaluates whether the technique allows to perform *browser fingerprinting*, cross-*browser fingerprinting*, or both. It also indicates if the technique writes information in the client system (Active) or not (Passive).

Also indicates if the technique relies on a comparison database to perform the *fingerprinting*.

Summary

This chapter depicted the techniques of *web-based fingerprinting* currently known and showed how to put them to practice using the technologies described in Chapter 2. References were made to previous works that depicted these techniques.

Whenever possible, examples of the technique's outputted information were collected and included. Special attention was given to showing the different behaviours of browsers when faced with the same scripts. The advantages and disadvantages inherent to the application of these techniques were weighted from the *fingerprinter*'s standpoint.

The "Taxonomy" section compared the various techniques by their ability to perform *browser fingerprinting* or *cross-device fingerprinting*, whether they use passive or active data collection and whether the techniques need a comparison database to support the *fingerprinting* process.

4 Threats of *web-based fingerprinting*

This chapter will discuss the threats that web-based fingerprinting presents both to security of systems and privacy of system users.

The legitimacy of this processing will also be verified considering the current legal framework.

4.1 Threats to privacy

Due to its almost completely unnoticeable nature, *web-based fingerprinting* raises several privacy issues. The obvious one is the possibility to track a user's online behaviour without his consent. Although some may argue that the identification of the user himself is never known and, therefore, the tracking will always be anonymized, this is not a lasting corollary.

While initially, the user's identity is unknown, each collection of information shortens the universe of possible individuals. The more data is collected, the narrower will the possible set be and more identifiable the individual gets.

As stated by the Article 29 Data Protection Working Party on the *Opinion 9/2014 on the Application of Directive 2002/58/EC to Device fingerprinting*:

> *(…) a fingerprint provides the ability to distinguish one device from another and can be used as a covert alternative for cookies to track internet behaviour over time. As a result, an individual may be associated, and therefore identified, or made identifiable, by that device fingerprint.*

The risks to privacy increases even more because *the unique set of information elements is not only available to the website publisher, but also to many other third parties*.

4.1.1 EU Legal framework regarding *fingerprinting*

Within the European Union, the Directive 2002/58/EC (European Parliament and the Council of the European Union, 2002) applies to the data processings that take place with the use of electronic communications.

Article 5(3) of Directive 2002/58/EC, as amended by Directive 2009/136/EC (European Parliament and the Council of the European Union, 2009), reinforced the protection of users of electronic communication networks and services by requiring informed consent before information is stored or accessed in the user's (or subscriber's) terminal device. The requirement applies to all types of information stored or accessed in the user's terminal device, including cookies and similar mechanisms.

Although the Article 5(3) requires an explicit consent from the user for storing and reading of data from his system, it also states that:

> *This shall not prevent any technical storage or access for the sole purpose of carrying out or facilitating the transmission of a communication over an electronic communications network, or as strictly necessary in order to provide an information society service explicitly requested by the subscriber or user.*

In summary, *fingerprinter*s have two options: to ask for the user's consent to carry on the *fingerprinting* process, while providing the user with information about the *fingerprinting* and describing the actors involved in the processing or, in alternative, not to ask for user's consent and to justify to the authorities that *fingerprinting* is needed to facilitate the transmission of a communication or, strictly necessary to provide a service requested by the user.

Although the decision is for the *fingerprinter*s to make, it's fair to say that none of the options suits the data collectors. The request for consent would invalidate the silent approach, which is passive *fingerprinting*'s biggest advantage. Advertisers, or data collectors that supply data to advertisers, would rather prefer to collect information of web users in a stealth and quiet way, avoiding problematical questions about users' rights to access their own data or what third-parties will be able to access that data.

Let's face it, how many of us would agree to proceed to visit a page knowing that a unique signature would be extracted from our system, in order to track further accesses? A positive answer to such request would be comprehensible in specific a scenario where the user would get some benefits from being identified by a website. For example, a website that would compare the fingerprint of all devices logging in order to verify it they would belong to the user, and requiring a second authentication whenever the device would not be recognized. In such a scenario consent makes sense, because the universe of users is limited and the *fingerprinting* is done for security reasons.

Not many more scenarios come to mind where users would gladly consent such an intrusive processing.

This leaves *fingerprinter*s with the second option: prove that the processing is needed to provide the service. This seems more like a "no option" since, at first glance, *fingerprinting* is not

necessary to provide service to web users (unless we're talking about a security service similar to the one above, where *fingerprinting* would have an important role).

Bottom-line, the way that most websites implement *fingerprinting* techniques nowadays is not legal under the EU law.

4.1.2 Cookies and *web-based fingerprinting*

The Article 29's decision on *device fingerprinting* (Article 29 Data Protection Working Party, 2014) states the following about *fingerprinting* and cookies:

> *In contrast to HTTP cookies, device fingerprinting can operate covertly. There are no simple means for users to prevent the activity and there are limited opportunities available to reset or modify any information elements being used to generate the fingerprint. As a result, device fingerprints can be used by third-parties to secretly identify or single out users with the potential to target content or otherwise treat them differently.*

Unlike cookies, most *fingerprinting* techniques can be completely silent, not leaving any trace once the data processing is over. This is where the true strength (and threats!) of *fingerprinting* lies - in its ability to operate covertly.

Regarding cookies, the Article 5(3) of the ePrivacy Directive (i.e. Directive 2002/58/EC, amended by the Directive 2009/136/EC) states that websites must require prior informed consent for storage or access to information stored on a user's terminal equipment. In other words, websites using cookies must ask visitors for their consent before they can install or read cookies.

Although the directive was first passed in 2009, many countries struggled to make the federal rule mesh with their own local implementation and privacy laws and, by the time the original deadline for adoption arrived in 2011 only Denmark and Estonia had enacted national laws that were deemed compliant[45].

Currently European countries are implementing the regulation on their internal laws and, at certain point, website owners will have to decide what to do about cookies: to comply with the regulation or to abandon cookies and go with an alternative technology that won't make them ask for user consent.

---

[45] The 2011 article "Cookies law: Only two EU states implement full measures – so far" from The Register sums up the situation at the time. Available at http://www.theregister.co.uk/2011/05/25/european_commission_cookies_directive/

Although, *web-based fingerprinting* is also bound to ask for consent according to the ePrivacy Directive, there is the risk that site owners might understand that the legislation is only applied to HTTP cookies, using this technology as a substitute for cookies.

4.2 Threats to security

Aside from being a threat to privacy, *web-based fingerprinting* can also be a threat to the system security of the clients.

In its essence, *fingerprinting* is all about collecting information from the systems. The more specific, the better. Software (e.g. browser and plugin) versions and operating system releases are security-related information that can be retrieved by *fingerprinting*.

Whoever has access to that information might be able to identify unpatched software or operating systems out of support (e.g. Microsoft Windows XP), creating an opportunity window to exploit the system.

This is the basic premise of the *Blackhole Exploit Kit* (BHEK).

BHEK is an exploit kit, created in 2010, designed to identify software vulnerabilities in client machines communicating with it and exploiting discovered vulnerabilities to upload and execute malicious code on the client. The developer of computer security software, Sophos, published a report regarding the distribution of web threads in 2012. In that report (see Howard, 2012), web attacks performed with aid of BHEK represented 22% of the total attacks registered in 2012.

The most common way hackers spread this exploit kit is by compromising legitimate websites and servers to serve the code for them. The next step is to get the user's browser to load the served code from what is commonly called as the 'landing page' of the exploit kit. According to Howard:

> *The purpose of the landing page is straightforward: to fingerprint the machine.*
>
> *The landing page used by Blackhole uses code from the legitimate PluginDetect library to identify: OS, Browser (and browser version), Adobe Flash version, Adobe Reader version, Java version.*

After the *fingerprinting*, the landing page will load one or more exploits specifically targeting the victim's browser and version. The vulnerabilities are targeted based on the potential vulnerabilities that may be present.

This shows how serious *fingerprinting* is, from a security perspective, and the risks that it entangles.

Summary

This chapter displayed the threats to privacy and security inherent to the processes of *fingerprinting* and tracking devices.

Currently, websites using *web-based fingerprinting* without user consent are clearly in transgression of the law, unless their owners prove that such processing is needed for providing service to the user.

It seems plausible to expect that site owners will try to find alternative mechanisms to HTTP cookies, once faced will the need to abide to the ePrivacy Directive. Whether they will turn towards *web-based fingerprinting* or not is still to be seen, but the fact is that this technology is the one that most resembles HTTP cookies.

On the security perspective, having a detailed collection of software versions and releases from one's system represents a serious threat for the end user. *Fingerprinting* has proven to be a tool for malware kits like BHEK.

Since information collection of the victim's system is the first phase of any attack, *web-based fingerprinting* represents precious tool for attackers.

5 Mitigations to *web-based fingerprinting*

As shown in the previous chapter, privacy and system security of web users are at stake whenever their Internet browsing activity is subjected to *fingerprinting* techniques.

Users can't be completely sure that websites are not *fingerprinting* them. Even in a scenario where the source code is available on the client side and the user is familiar with programming languages, the frequent use of obfuscation techniques on the *fingerprinting* scripts might render the code too difficult to interpret (or make the reverting process too cumbersome).

This chapter will evaluate possible mitigations to *web-based fingerprinting*. In the current context, the term "mitigation" it used to refer to any technique or technology that could be put to practice by an web user to prevent or reduce the risk of being identified or singularized through the use of *fingerprinting* techniques.

It's important to stretch that the objective of this exercise is to identify and evaluate measures accessible to the ordinary web user, leaving out any solutions that would involve high technological expertise or a great investment of time and/or resources. The evaluation of the different approaches will be done in an academic perspective, meaning that this work won't propose a particular solution.

Whenever any of the measures has already been proposed previously by other authors, a reference will be done to the work in question.

5.1 Script blocking

Since most of *fingerprinting* process actually takes place in the client side, a possible countermeasure would be to configure the browser to block the execution of scripts, and request permission to run scripts on demand.

The browser would build a list of scripting-allowed sites, based on the user's preferences, reducing the impact of the script blocking in the user's experience. Browser plugins such as

NoScript [46]or *Adblock Plus*[47] allow managing the list of allowed sites to run scripts on the browser.

Evaluation: Because there is a high number of websites that use them, the blocking of scripts is an extremely restrictive measure that highly upsets the user experience. The use of script blocking plugins helps levering the usability, by automating the process of authorization to known pages. Eckersley (in Eckersley, 2010) stated that *NoScript is a useful privacy enhancing technology that seems to reduce fingerprintability*.

However, Mowery, et al. show that script blocking plugins like *NoScript* can be exploited to provide a fingerprint themselves (Mowery, et al. 2011). The authors take advantage of the behaviour of the plugin: every requested page has its scripts checked and any scripts from other domains included in that page. By creating an extensive list of websites that users might have whitelisted (because they trust them), the *fingerprinter*s can add script calls to these domains and register the browser's behaviour. For every domain whose script is allowed to run, the *fingerprinter* adds one more element to the signature. With a list long enough, a unique signature will eventually be produced.

It is arguable to consider *NoScript* such a widespread and common plugin that would make *fingerprinter*s want to create such a specific mechanism.

5.2 Using a "standard" browser

One of the "self-defence" measures proposed by the *Panopticlick* website is the use of a *"standard", "common" browser*. The website states that:

> *Current versions of the iPhone, Android, and Blackberries do not vary much with respect to plugins, installed fonts, or screen size. This situation may well change in the future, but until it does, most of these devices are far less fingerprintable than any sort of desktop PC.*

The idea is that the more the browser blends with the others, the more difficult it will be to fingerprint.

Evaluation: While it is true that some uncommon browsers may be easier to spot, because their behaviour might not be within the threshold of the most popular browsers, it is also true that

[46] NoScript webpage available at https://noscript.net
[47] Adblock Plus webpage available at https://adblockplus.org

there is an ever-growing universe of browsers branching from the standard browsers (e.g. *Waterfox*, *Pale Moon*, *Icecat* and *Wyzo* are examples of browsers modified from the original *Mozilla Firefox*). These browsers might show a behaviour so similar to the original ones that a *fingerprinter* might not notice the difference.

On another hand, many *fingerprinting* techniques abstract themselves from the browser, focusing on the system. The *fingerprinting* by operating system features, hardware features or font detection are examples of techniques that would retrieve the system information, regardless of the browser type.

5.3 Modified browser

At the time of the present work, there is no known browser software publicly available on the market that fully addresses (and tackles) the privacy risks inherent to *web-based fingerprinting*.

In a certain way, it is understandable why none of the main browser vendors would decide to include *fingerprinting* protection in their products. To be efficient at avoiding *fingerprinting*, a browser would have to show an irregular behaviour, constantly sending different information about the system during the browsing time, making it impossible for the *fingerprinter* to associate different accesses to a certain device. Additionally, plugins such as Adobe Flash and Java, that could circumvent the browser privacy policy, would have to be disabled or, at least, have their ability to retrieve system information inhibited. Finally, all client side scripting should be disabled in order to avoid any API call that could access system information that would allow singularization of the device.

It easy to foresee that such mitigations would have a noticeable effect in the user's browsing experience, limiting his ability to access media contents and blocking any dynamic behaviour scripted in webpages. For most users this scenario would be unacceptable as functionality would be terribly restricted, which explains the lack of interest from browser vendors.

Therefore, any current approach to *web-based fingerprinting* mitigation through browser limitations will, necessarily, have to be done by creation of a new browser or modification of an existing browser (preferably an open-source one).

5.3.1 PriVaricator

In the work of Nikiforakis, et al. (2014), the authors propose a "solution to the problem of browser-based *fingerprinting*", by customizing the behaviour of a browser.

The premise for the team was that "the real problem with *fingerprinting* is not uniqueness of a fingerprint, it is linkability", therefore the efforts made were directed at modifying the browser to make every visit appear different to a *fingerprinting* site.

Of the multiple *fingerprinting* techniques the authors found indispensable to address the Plugin Enumeration (previously shown in Section 3.3.1) and the Screen Properties (Section 3.6.1). Apparently the canvas-based *fingerprinting* was considered but left out because it did not appeared to be "widely used" at the time. Therefore, the modified *Chromium* browser was tuned to intercept the accesses to DOM properties and change the values returned to the JavaScript environment. Screen offset (width and height) values would be changed between to a random value between a defined range, avoiding to provide noticeable unrealistic values that would render the process futile. For the plugins list, every plugin would have a random number associated, created at every access to this property. Depending on the random value, the plugin would or would not be returned by the browser.

According to the team, four different *fingerprinting* providers'[48] websites were visited 1.331 times each and the results were registered.

In the JavaScript font-based detection technique, for instance, it was shown that when the "lying threshold" was higher (i.e. the browser was lying too much), the *fingerprinter* considered the fonts as "present". This reassured the idea that diverging too much from the original information could prove to be detrimental for the deceiving purpose.

Additionally, it was shown that certain *fingerprinter*s still managed to retrieve unique signatures of the *PriVaricator* accesses, what demonstrated that "most *fingerprinting* providers derive a fingerprint by following a more complicated approach than just hashing all *fingerprintable* attributes together."

Evaluation: This approach tries to balance the mitigation of *fingerprinting* with browser usability. The authors were clear about showing that the objective was not to address all types of *web-based fingerprinting*, as it would have a cost on the browser performance and on the end user experience (e.g. blocking of media content, blocking of dynamic page behaviour, etc.).

The customized browser managed to avoid *fingerprinting* for a fair amount of websites with a small number of effects over the presentation of the webpages.

However, addressing only JavaScript based techniques like font/plugin enumeration and screen resolution is a simplistic approach to *fingerprinting*. As shown throughout this work, there is a

[48] BlueCava, PetPortal, Coinbase and fingerprintjs

myriad of other techniques that were left aside from the *PriVaricator* approach, rendering it still *fingerprintable*.

Weighting positive and negative elements, the *PriVaricator* is still the best measure proposed to counter *web-based fingerprinting* to date.

5.3.2 Tor Browser

Tor Browser consists in a modified *Mozilla Firefox ESR* web browser and is part of a bundle that uses the Tor network. It includes the following countermeasures to *fingerprinting*: *NoScript* extension to ensure no scripts bypass the anonymity of the browser; prompting before returning canvas image; *WebGl* contents disabled by default, limitation of number of fonts requested per page and mitigation of the User Agent information. The Tor Browser development team recommends users to avoid contents like Flash or Adobe Acrobat, also to ensure anonymity.

Evaluation: Some authors refer that *Tor Browser* showed good resilience to a few *fingerprinting* techniques. In the work Acar, et al. (2014), the authors state the following about mitigations to canvas *fingerprinting*:

> *(…) asking user permission for each canvas read attempt may be the only effective solution. Indeed, this is precisely the technique adopted in the Tor Browser, the only software that we found to successfully protect against canvas fingerprinting. Specifically, the Tor Browser returns an empty image from all the canvas functions that can be used to read image data.*
>
> *(…)As for more traditional fingerprinting techniques, the Tor browser again appears to be the only effective tool. With the exception of a recent Mozilla effort to limit plugin enumeration.*

In another work (Acar, 2014) the author refers that:

> *(…) the Tor Browser is the only piece of software that adopts a well informed and carefully designed approach based on obfuscation to thwart fingerprinting.*

Although the overall opinions are favourable to *Tor Browser's* mechanisms to address *fingerprinting*, it's important to remember that the use of Tor networks adds specific constraints and risks to a network connection. For one side, the time needed to access contents is greatly increased, because of the relay times inherent to the onion routing. This factor alone would be enough to deter most users, who would not accept gladly such a decrease of performance in their network access.

On other side, the fact that Tor cannot encrypt the traffic between an exit node and the target, requiring end-to-end SSL or TLS to ensure encryption, creates a new anonymity problem. Users must be aware about the risk of eavesdropping inherent to this technology and decide accordingly.

Summary

Web-based fingerprinting is known to be difficult to avoid. Most of the existing or proposed measures to mitigate *web-based fingerprinting* require users to limit their browsing experience. These limitations can take the form of customized browsers, which would confine the execution of scripts and plugins to a minimum, or the simple blocking of scripts, ideally with an option to add trusted sites.

A few less-harming limitations have been proposed, like the *PriVaricator* browser, but are still not available for users.

One thing is certain, the sanitization of the browser's output is not enough to ensure the mitigation of *fingerprinting*, because there are techniques based in browser's performance that will not rely in the information handed by the browser.

6 Conclusion

The current work depicted a study of the presently known *web-based fingerprinting* techniques. A special effort was put into gathering the biggest number of different techniques, to make of this study the most extensive depiction of *web-based fingerprinting* techniques. During the collecting of techniques it became obvious that some have more ambitious goals than others. While some aim simply at identifying the browser brand (such as "browser rounding and fractional pixels"), others like "Font enumeration" are designed to create a signature almost by themselves. Still, techniques that might seem limited, in the sense that they don't provide a definitive signature, can always be linked with others, providing more entropy to the data set.

The collected techniques were categorized in a taxonomy that will allow readers to easily identify different types of techniques, their capabilities, limitations and similarities. The categories were defined bearing in mind that new techniques might appear in the future and still have a category where they will fit in. The process of category isolation (i.e. preventing categories from overlapping each other) was one of the most time consuming endeavours in this work, mainly because some techniques spanned across multiple categories.

The websites listed in Section 2.4 (Websites for browser testing) provided to be very helpful. Not only for testing purposes and retrieval of examples, but also for providing valuable cues for additional *fingerprinting* techniques. Although the first laid work on *device fingerprinting* dates back to 2009, the literature about the subject is not too extensive. Because of this, websites presenting techniques, whether for helping *fingerprinter*s to code scripts or for awareness purposes, are an important contribution for the enrichment of the researcher's knowledge.

Moreover, the lawful implications of this type of processing were discussed, attending to the legal framework currently available.

At first glance, it seems that *web-based fingerprinting* will be a dead-end road for site owners trying to escape the ePrivacy Directive, because it also falls in the same category of HTTP cookies. Nevertheless, it's stealthily nature might be too appealing for those whose activity relies greatly in HTTP cookies nowadays, such as the advertisement sector.

Having the chance to obfuscate scripting code and using seemingly legitimate code to measure browser performance or other particular behaviour, *fingerprinter*s might have reasons to believe that they can "get away with it".

Unfortunately, they might be right.

Of course, obfuscated code can be reverted and programs can be designed to search for *fingerprinting* patterns in webpages. But this would be time-consuming (even when done automatically) and would result in an answer such as "The page has 75% probability of being *fingerprinting* your device. Proceed to page?". It's arguable if the common user would be willing to be prompted by these types of alerts. Wouldn't he/she blindly answer affirmatively just to proceed to the requested page?

Still, legitimate switching from cookies to *web-based fingerprinting* is possible. In fact, if the signature is trustable, a fingerprint can be as good as an HTTP cookie to manage user sessions, save preferences and even trace users across websites.

It seems that the answer to whether switching to *fingerprinting* or not will depend on how legitimate is the data processing made with the cookies nowadays. Websites that make use of cookies for the sole purpose of keeping the session alive won't have problems in informing the users of this mechanism.

Security concerns will always surround *device fingerprinting*, since the collection of software versions and architectural details is possible and known to be the first phase of an attack.

Sadly, the mitigation measures that exist today fall short for what is needed.

As more and more *fingerprinting* techniques appear, some argue that the only reasonable countermeasure is to block scripting. This seems to be an excessive measure, with high probability of having users bypassing the security mechanism just to access the desired contents.

Proposed methods like the *PriVaricator* browser are good approaches but, like was said in the previous chapter, it lacks protection for some other *web-based fingerprinting* techniques, rendering it flawed.

It does not seem reasonable to hope for a single solution that would render *fingerprinting* techniques useless, as they are too many and too diverse. That's why the current work referred to "mitigations" and not "solutions".

Fingerprinting technologies are constantly evolving. It's important that browser manufacturers adapt their products to be less exposed to these techniques. A good starting point would be for all cached objects (HTTP, Flash and HTML5 cookies) to be managed centrally, for user convenience, for example.

In conclusion, web-based fingerprinting is a powerful tool. Its strength lies in the fact that browsers nowadays are not mere displayers of webpages. They have evolved to provide a better user experience: by hosting ecosystems of plugins, extensions and add-ons, and by providing APIs for more functionality. Users demand faster browsers but, at the same time, compatibility with the most popular and recent media formats and programming languages.

Manufacturers provide the products with said requirements, but the problem is that browsers get more noisy and talkative as they become more compatible with other technologies. Talkative technologies are a big help for *fingerprinters in general*.

The *Web-based fingerprinting* concept emerged after the realization that a device becomes unique because it is used by a unique entity – in this case, a human being. In one way or another, the human influence over a device will, eventually, shape it to be more user-like, making it unique.

Because uniqueness is a highly prized characteristic in human behaviour, devices will keep shaping together with users, into more individualized forms and configurations. Maybe, making *fingerprinting* easier than it is today.

Acronyms

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CCTV	Closed-circuit television
CLI	Common Language Infrastructure
DNT	Do Not Track
DPA	(National) Data Protection Authority
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
ISP	Acronym for Internet service provider
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extension
NAT	Network Address Translation
OS	Operating System
VPN	Virtual Private Network
W3C	Acronym for World Wide Web Consortium
WebGL	Web Graphics Library
WebRTC	Web Real-Time Communication

Glossary

CONSENT - A freely given specific and informed indication of his or her wishes by which the individual signifies his or her agreement to a certain data processing.

DATA CONTROLLER -Institution or body that determines the purposes and means of the processing of personal data. The data controller is also responsible for the security measures put in place to protect data.

DATA PROCESSOR – Agent that performs an action over a piece of data. Access, insertion, modification, deletion, transmission and correlation of data are all possible actions of a data processor.

DATA SUBJECT - The data subject is the person whose personal data are collected, held or processed.

DIRECTIVE (EU) - A form of legislation that is destined to the Member States. It sets out the objective or policy which needs to be attained. The Member States must then pass the relevant domestic legislation to give effect to the terms of the Directive within a time frame set in the directive.

FINGERPRINT – A combination of unique characteristics associated to an individual that make possible the univocal identification of that individual from the remaining universe.

FINGERPRINTER – Agent who carries on the process of collecting, storing and matching fingerprints with the existing ones.

SIGNATURE – See FINGERPRINT.

SINGULARIZATION – Process that allows a data processor to distinguish a single individual from a certain universe, without identifying him, based on the unique set of features that he processes.

Bibliography

Acar, G. (2014). Obfuscation for and against device fingerprinting Position Paper for Symposium on Obfuscation New York University, February 15, 2014

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

Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., & Preneel, B. (2013). FPDetective: Dusting the Web for Fingerprinters Categories and Subject Descriptors. CCS '13 (20th ACM Conference on Computer and Communications Security).

Alsenoy, B., Verdoodt, V., Heyman, R., Ausloos, J. & Wauters, E. (2015). From social media service to advertising network - A critical analysis of Facebook's Revised Policies and Terms, Interdisciplinary Centre for Law and ICT/Centre for Intellectual Property Rights (ICRI/CIR) of KU Leuven, Draft 23/02/2015.

Anbar, W. (2011, May 12). Your privacy and Adobe Flash Player. *Adobe.com*. Retrieved from http://www.adobe.com/devnet/flashplayer/articles/privacy.html .

Article 29 Data Protection Working Party, (2014), Opinion 9/2014 on the Application of Directive 2002/58/EC to Device fingerprinting, 14/EN, WP 224 (adopted on 25 November 2014).

Boda, K., Földes, Á. M., Gulyás, G. G., & Imre, S. (2012). User tracking on the web via cross-browser fingerprinting. In Information Security Technology for Applications (pp. 31-46). Springer Berlin Heidelberg.

Brod (2012, April 2). Mac Flashback Exploiting Unpatched Java Vulnerability. *F-Secure*. Retrieved from https://www.f-secure.com/weblog/archives/00002341.html

Constantin, L. (2013a, Jan 15). Java exploit used in Red October cyberespionage attacks, researchers say. *ComputerWorld*. Retrieved from http://www.computerworld.com/article/2494233/malware-vulnerabilities/java-exploit-used-in-red-october-cyberespionage-attacks--researchers-say.html

Constantin, L. (2013b, March 4). Oracle releases emergency fix for Java zero-day exploit. *PCWorld*. Retrieved from http://www.pcworld.com/article/2030056/oracle-releases-emergency-fix-for-java-zeroday-exploit.html

Das, A., Borisov, N., & Caesar, M. (2014). Fingerprinting Smart Devices Through Embedded Acoustic Components. doi:10.1145/2660267.2660325

Eckersley, P. (2010). How unique is your web browser? Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6205 LNCS, 1–18. doi:10.1007/978-3-642-14527-8_1

European Parliament and the Council of the European Union. (1995). Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Official Journal of the EC, 23(6).

European Parliament and the Council of the European Union. (2002). Directive 2002/58/EC of the European Parliament and of the Council of 12 July 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector (Directive on privacy and electronic communications) (L 201. Official Journal of the European Communities, 37–47.

European Parliament and the Council of the European Union. (2009). Directive 2009/136/EC of the European Parliament and of the Council of 25 November 2009 amending Directive 2002/22/EC on universal service and users' rights relating to electronic communications networks and services, Directive 2002/58/EC concerning the processing of personal data and the protection of privacy in the electronic communications sector and Regulation (EC) No 2006/2004 on cooperation between national authorities responsible for the enforcement of consumer protection laws. (L 337. Official Journal of the European Union, 11-36)

Gerds, E., (2015, April 13), PluginDetect, Retrieved from http://www.pinlady.net/PluginDetect/

Goodin, D. (2012, Oct 18). Apple removes Java from all OS X Web browsers. *Arstechnica*. Retrieved October 26, 2014, from http://arstechnica.com/apple/2012/10/apple-removes-java-from-all-os-x-web-browsers

Goodin, D. (2013a, Jan 28). Java's new "very high" security mode can't protect you from malware. *Arstechnica*. Retrieved from http://arstechnica.com/security/2013/01/javas-new-very-high-security-mode-cant-protect-you-from-malware

Goodin, D. (2013b, Jan 31). Firefox to block content based on Java, Reader, and Silverlight. *Arstechnica*. Retrieved from http://arstechnica.com/security/2013/01/firefox-to-block-content-based-on-java-reader-and-silverlight/

Hasse, J., Gloe, T., & Beck, M. (2013). Forensic Identification of GSM Mobile Phones. ACM Workshop on Information Hiding and Multimedia Security, 131–140. doi:10.1145/2482513.2482529

Howard, F. (2012). Exploring the Blackhole exploit kit. Sophos Technical Paper

Janc, A., & Olejnik, L. (2010). Web browser history detection as a real-world privacy threat. In Computer Security–ESORICS 2010 (pp. 215-231). Springer Berlin Heidelberg

Kilgour, A. (2014, November 25), Browser Rounding and Fractional Pixels

Kohno, T., Broido, A., Claffy, K.C. (2005). Remote physical device fingerprinting, IEEE Transactions on Dependable and Secure Computing, 2(2), April-June 2005

Mayer, J. (2009). Any person... a pamphleteer: Internet Anonymity in the Age of Web 2.0. Undergraduate Senior Thesis, Princeton University

Mayer, J. R., & Mitchell, J. C. (2012). Third-party web tracking: Policy and technology. Proceedings - IEEE Symposium on Security and Privacy, 413–427. http://doi.org/10.1109/SP.2012.47

Mowery, K., Bogenreif, D., Yilek, S., & Shacham, H. (2011). Fingerprinting information in JavaScript implementations. Proceedings of W2SP, 2.

Mowery, K., & Shacham, H. (2012). Pixel Perfect: Fingerprinting Canvas in HTML5. Web 2.0 Security & Privacy 20 (W2SP), 1–12.

Neumann, C., Heen, O., & Onno, S. (2012). An empirical study of passive 802.11 *device* fingerprinting. Proceedings - 32nd IEEE International Conference on Distributed Computing Systems Workshops, ICDCSW 2012, 593–602. doi:10.1109/ICDCSW.2012.8

Nikiforakis, N., Acar, G., (2014). Browse at your own risk. Spectrum, IEEE, vol.51, no.8, pp.30, 35, August 2014

Nikiforakis, N., Joosen, W., & Livshits, B. (2014). PriVaricator: Deceiving Fingerprinters with Little White Lies. Research.Microsoft.Com.

Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., & Vigna, G. (2013). Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. Proceedings - IEEE Symposium on Security and Privacy, 541–555. doi:10.1109/SP.2013.43

Perry, M., Clark, E., Murdoch & S. (2015, May 06). The Design and Implementation of the Tor Browser. *TorProject*. Retrieved from https://www.torproject.org/projects/torbrowser/design/#*fingerprinting*-linkability

Olejnik, L., Castelluccia, C., & Janc, A. (2012). Why Johnny can't browse in peace: On the uniqueness of web browsing history patterns. In 5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2012).

Oliver, B. J., Cheng, S., Manly, L., Zhu, J., Dela Paz, R., Sioting, S., & Leopando, J. (2012). Blackhole Exploit Kit: A Spam Campaign, Not a Series of Individual Spam Runs, Trend Micro Incorporated Research Paper. 1–19.

Speers, R., Goodspeed, T., Jenkins, I.R., Shapiro, R. & Bratus, S., (2014), Fingerprinting IEEE 802.15.4 Devices with Commodity Radios. Dartmouth Computer Science Technical Report TR2014-746, March 2014.

Stöber, T., Frank, M., Schmitt, J., & Martinovic, I. (2013). Who do you sync you are?: smartphone fingerprinting via application behaviour. Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks - WiSec '13, 7. doi:10.1145/2462096.2462099

Tirtea, R., Castelluccia, C. & Ikonomou, D. (2010). Bittersweet Cookies: Some Security and Privacy Considerations. ENISA (European Network and Information Security Agency)

Zakas, N. (2010, October 13,). How many users have JavaScript disabled? *Yahoo.com*. Retrieved from https://developer.yahoo.com/blogs/ydn/many-users-javascript-disabled-14121.html

Zalewski, M. (2012). The Tangled Web: A Guide to Securing Modern Web Applications. No Starch Press.

Appendices

Appendix I – Example of a browser plugins and mime-types list, retrieved with JavaScript

(Information retrieved from the website *Panopticlick* for the

Mozilla Firefox browser)

Plugin 0: Adobe Acrobat; Adobe PDF Plug-In For Firefox and Netscape 11.0.11; nppdf32.dll; (Acrobat Portable Document Format; application/pdf; pdf) (Adobe PDF in XML Format; application/vnd.adobe.pdfxml; pdfxml) (Adobe PDF in XML Format; application/vnd.adobe.x-mars; mars) (Acrobat Forms Data Format; application/vnd.fdf; fdf) (XML Version of Acrobat Forms Data Format; application/vnd.adobe.xfdf; xfdf) (Acrobat XML Data Package; application/vnd.adobe.xdp+xml; xdp) (Adobe FormFlow99 Data File; application/vnd.adobe.xfd+xml; xfd).

Plugin 2: Google Update; Google Update; npGoogleUpdate3.dll; (; application/x-vnd.google.update3webcontrol.3;) (; application/x-vnd.google.oneclickctrl.9;).

Plugin 3: Java Deployment Toolkit 7.0.450.18; NPRuntime Script Plug-in Library for Java(TM) Deploy; npdeployJava1.dll; (; application/java-deployment-toolkit;).

Plugin 4: Java(TM) Platform SE 7 U45; Next Generation Java Plug-in 10.45.2 for Mozilla browsers; npjp2.dll; (Java Applet; application/x-java-applet;) (JavaBeans; application/x-java-bean;) (; application/x-java-vm;) (; application/x-java-applet;version=1.1.1;) (; application/x-java-bean;version=1.1.1;) (; application/x-java-applet;version=1.1;) (; application/x-java-bean;version=1.1;) (; application/x-java-applet;version=1.2;) (; application/x-java-bean;version=1.2;) (; application/x-java-applet;version=1.1.3;) (; application/x-java-bean;version=1.1.3;) (; application/x-java-applet;version=1.1.2;) (; application/x-java-bean;version=1.1.2;) (; application/x-java-applet;version=1.3;) (; application/x-java-bean;version=1.3;) (; application/x-java-applet;version=1.2.2;) (; application/x-java-bean;version=1.2.2;) (; application/x-java-applet;version=1.2.1;) (; application/x-java-bean;version=1.2.1;) (; application/x-java-applet;version=1.3.1;) (; application/x-java-bean;version=1.3.1;) (; application/x-java-applet;version=1.4;) (; application/x-java-bean;version=1.4;) (; application/x-java-applet;version=1.4.1;) (; application/x-java-bean;version=1.4.1;) (; application/x-java-applet;version=1.4.2;) (; application/x-java-bean;version=1.4.2;) (; application/x-java-applet;version=1.5;) (; application/x-java-bean;version=1.5;) (; application/x-java-applet;version=1.6;) (; application/x-java-bean;version=1.6;) (; application/x-java-applet;version=1.7;) (; application/x-java-bean;version=1.7;) (; application/x-java-applet;jpi-version=1.7.0_45;) (; application/x-java-bean;jpi-version=1.7.0_45;) (; application/x-java-vm-npruntime;) (;

application/x-java-applet;deploy=10.45.2;) (; application/x-java-applet;javafx=2.2.45;).

Plugin 5: Microsoft Office 2013; The plugin allows you to have a better experience with Microsoft Lync; npMeetingJoinPluginOC.dll; (Lync Plug-in for Firefox; application/vnd.microsoft.communicator.ocsmeeting;).

Plugin 6: Microsoft Office 2013; The plugin allows you to have a better experience with Microsoft SharePoint; NPSPWRAP.DLL; (SharePoint Plug-in for Firefox; application/x-sharepoint;) (; application/x-sharepoint-uc;).

Plugin 7: PDF Architect 2; PDF Architect 2; np-previewer.dll; (PDF File; application/pdf; pdf) (WWF File; application/wwf; wwf).

Plugin 8: Photo Gallery; NPWLPG; NPWLPG.dll; (Photo Gallery; application/x-wlpg3-detect; wlpg) (Photo Gallery; application/x-wlpg-detect; wlpg).

Plugin 9: Shockwave Flash; Shockwave Flash 17.0 r0; NPSWF32_17_0_0_169.dll; (Adobe Flash movie; application/x-shockwave-flash; swf) (FutureSplash movie; application/futuresplash; spl).

Plugin 10: Silverlight Plug-In; 5.1.40416.0; npctrl.dll; (npctrl; application/x-silverlight; scr) (; application/x-silverlight-2;).

Plugin 11: VLC Web Plugin; VLC media player Web Plugin; npvlc.dll; (MPEG audio; audio/mpeg; mp2,mp3,mpga,mpega) (MPEG audio; audio/x-mpeg; mp2,mp3,mpga,mpega) (MPEG video; video/mpeg; mpg,mpeg,mpe) (MPEG video; video/x-mpeg; mpg,mpeg,mpe) (MPEG video; video/mpeg-system; mpg,mpeg,mpe,vob) (MPEG video; video/x-mpeg-system; mpg,mpeg,mpe,vob) (MPEG-4 audio; audio/mp4; aac,mp4,mpg4) (MPEG-4 audio; audio/x-m4a; m4a) (MPEG-4 video; video/mp4; mp4,mpg4) (MPEG-4 video; application/mpeg4-iod; mp4,mpg4) (MPEG-4 video; application/mpeg4-muxcodetable; mp4,mpg4) (MPEG-4 video; video/x-m4v; m4v) (AVI video; video/x-msvideo; avi) (Ogg stream; application/ogg; ogg) (Ogg video; video/ogg; ogv) (Ogg stream; application/x-ogg; ogg) (VLC plug-in; application/x-vlc-plugin;) (Windows Media Video; video/x-ms-asf-plugin; asf,asx) (Windows Media Video; video/x-ms-asf; asf,asx) (Windows Media; application/x-mplayer2;) (Windows Media; video/x-ms-wmv; wmv) (Windows Media Video; video/x-ms-wvx; wvx) (Windows Media Audio; audio/x-ms-wma; wma) (Google VLC plug-in; application/x-google-vlc-plugin;) (WAV audio; audio/wav; wav) (WAV audio; audio/x-wav; wav) (3GPP audio; audio/3gpp; 3gp,3gpp) (3GPP video; video/3gpp; 3gp,3gpp) (3GPP2 audio; audio/3gpp2; 3g2,3gpp2) (3GPP2 video; video/3gpp2; 3g2,3gpp2) (DivX video; video/divx; divx) (FLV video; video/flv; flv) (FLV video; video/x-flv; flv) (Matroska video; application/x-matroska; mkv) (Matroska video; video/x-matroska; mkv) (Matroska audio; audio/x-matroska; mka) (Playlist xspf; application/xspf+xml; xspf) (MPEG audio; audio/x-mpegurl; m3u) (WebM video; video/webm; webm) (WebM audio;

audio/webm; webm) (Real Media File; application/vnd.rn-realmedia; rm) (Real Media Audio; audio/x-realaudio; ra) (AMR audio; audio/amr; amr) (FLAC audio; audio/x-flac; flac).

Plugin 12: VMware Remote Console Plug-in; VMware Remote Console Plug-in; np-vmware-vmrc.dll; (VMware Remote Console Plug-in; application/x-vmware-remote-console-2012;).

Appendix II – Example of JavaScript Performance Benchmarkings

(Information retrieved from the website *V8 Benchmark Suite* for the

Mozilla Firefox browser)

Test #1	Test #2	Test #3
Score: 6177	**Score: 6854**	**Score: 7025**
Richards: 5790	Richards: 6023	Richards: 6051
DeltaBlue: 10731	DeltaBlue: 20556	DeltaBlue: 13052
Crypto: 4957	Crypto: 4556	Crypto: 7585
RayTrace: 7356	RayTrace: 7027	RayTrace: 11904
EarleyBoyer: 11948	EarleyBoyer: 8513	EarleyBoyer: 18541
RegExp: 1852	RegExp: 1575	RegExp: 1064
Splay: 7188	Splay: 12982	Splay: 7595
NavierStokes: 5883	NavierStokes: 7060	NavierStokes: 5550

Test #4	Test #5	Test #6
Score: 3301	**Score: 5732**	**Score: 2877**
Richards: 5871	Richards: 4497	Richards: 913
DeltaBlue: 1950	DeltaBlue: 4894	DeltaBlue: 5633
Crypto: 4404	Crypto: 6418	Crypto: 4189
RayTrace: 4992	RayTrace: 6215	RayTrace: 6581
EarleyBoyer: 6239	EarleyBoyer: 11946	EarleyBoyer: 5915
RegExp: 636	RegExp: 1707	RegExp: 714
Splay: 2564	Splay: 4066	Splay: 1434
NavierStokes: 5496	NavierStokes: 16011	NavierStokes: 5458

Test #7	Test #8	Test #9
Score: 3904	**Score: 4014**	**Score: 4193**
Richards: 5798	Richards: 5599	Richards: 5987
DeltaBlue: 5508	DeltaBlue: 5858	DeltaBlue: 6268
Crypto: 3882	Crypto: 4424	Crypto: 4452
RayTrace: 6573	RayTrace: 6607	RayTrace: 6240
EarleyBoyer: 5832	EarleyBoyer: 5875	EarleyBoyer: 5935
RegExp: 702	RegExp: 761	RegExp: 761
Splay: 2971	Splay: 2866	Splay: 3663
NavierStokes: 5442	NavierStokes: 5485	NavierStokes: 5534

Test #10	Test #11	Test #12
Score: 6993	**Score: 7521**	**Score: 5516**
Richards: 9588	Richards: 17948	Richards: 5835
DeltaBlue: 16926	DeltaBlue: 3782	DeltaBlue: 4926
Crypto: 5192	Crypto: 8240	Crypto: 5354
RayTrace: 13912	RayTrace: 24765	RayTrace: 18999
EarleyBoyer: 11800	EarleyBoyer: 9109	EarleyBoyer: 8478
RegExp: 1593	RegExp: 1674	RegExp: 949
Splay: 4280	Splay: 4819	Splay: 6633
NavierStokes: 6066	NavierStokes: 10061	NavierStokes: 5491

Richards, *DeltaBlue*, *Crypto*, *RayTrace*, *EarleyBoyer*, *RegExp*, *Splay* and *NavierStokes* are benchmarking scripts included in the *V8*.

(Information retrieved from the website *SunSpider 1.0.2 JavaScript Benchmark* for the *Mozilla Firefox* browser)

Test #1

```
Total:                    725.2ms +/- 32.3%
-----------------------------------------

  3d:                     128.8ms +/- 35.4%
    cube:                  49.9ms +/- 41.0%
    morph:                 21.9ms +/- 33.8%
    raytrace:              57.0ms +/- 40.4%

  access:                  52.3ms +/- 36.1%
    binary-trees:          11.7ms +/- 54.8%
    fannkuch:              21.6ms +/- 35.3%
    nbody:                 11.5ms +/- 36.7%
    nsieve:                 7.5ms +/- 34.0%

  bitops:                  28.5ms +/- 29.4%
    3bit-bits-in-byte:      3.0ms +/- 31.8%
    bits-in-byte:           9.0ms +/- 34.9%
    bitwise-and:            5.0ms +/- 26.1%
    nsieve-bits:           11.5ms +/- 32.4%

  controlflow:             5.4ms +/- 31.9%
    recursive:             5.4ms +/- 31.9%

  crypto:                  74.6ms +/- 61.5%
    aes:                   32.9ms +/- 71.0%
    md5:                   29.4ms +/- 106.4%
    sha1:                  12.3ms +/- 35.5%

  date:                    95.2ms +/- 41.2%
    format-tofte:          37.0ms +/- 37.1%
    format-xparb:          58.2ms +/- 45.2%

  math:                    49.4ms +/- 33.0%
    cordic:                10.7ms +/- 39.3%
    partial-sums:          32.2ms +/- 33.7%
    spectral-norm:          6.5ms +/- 49.0%

  regexp:                  31.2ms +/- 38.8%
    dna:                   31.2ms +/- 38.8%

  string:                 259.8ms +/- 28.3%
    base64:                26.2ms +/- 26.4%
    fasta:                 23.9ms +/- 30.0%
    tagcloud:              74.9ms +/- 35.5%
    unpack-code:           97.4ms +/- 36.1%
    validate-input:        37.4ms +/- 34.4%
```

Test #2

```
Total:                    373.6ms +/- 11.3%
-----------------------------------------

  3d:                      75.2ms +/- 17.4%
    cube:                  29.7ms +/- 37.9%
    morph:                 15.3ms +/- 24.8%
    raytrace:              30.2ms +/- 18.9%

  access:                  30.8ms +/- 21.8%
    binary-trees:           6.5ms +/- 71.9%
    fannkuch:              11.7ms +/- 18.4%
    nbody:                  6.1ms +/- 41.5%
    nsieve:                 6.5ms +/- 50.8%

  bitops:                  19.9ms +/- 39.8%
    3bit-bits-in-byte:     1.8ms +/- 61.5%
    bits-in-byte:          5.7ms +/- 41.8%
    bitwise-and:           4.1ms +/- 43.1%
    nsieve-bits:           8.3ms +/- 46.6%

  controlflow:             4.4ms +/- 48.5%
    recursive:             4.4ms +/- 48.5%

  crypto:                  29.0ms +/- 14.4%
    aes:                   14.1ms +/- 17.8%
    md5:                    8.5ms +/- 18.7%
    sha1:                   6.4ms +/- 19.8%

  date:                    40.1ms +/- 12.9%
    format-tofte:          16.7ms +/- 17.5%
    format-xparb:          23.4ms +/- 20.6%

  math:                    27.3ms +/- 18.0%
    cordic:                 9.1ms +/- 56.7%
    partial-sums:          15.3ms +/- 6.6%
    spectral-norm:          2.9ms +/- 7.8%

  regexp:                  14.3ms +/- 17.0%
    dna:                   14.3ms +/- 17.0%

  string:                 132.6ms +/- 16.3%
    base64:                13.8ms +/- 35.8%
    fasta:                 14.3ms +/- 21.5%
    tagcloud:              37.7ms +/- 12.0%
    unpack-code:           48.5ms +/- 24.2%
    validate-input:        18.3ms +/- 33.4%
```

Appendix III – Example of OS and browser features, retrieved with Flash Plugin

(Information retrieved from the website *BrowserLeaks.com* for the

Mozilla Firefox and *Google Chrome* browsers)

Shockwave Flash plugin details

Mozilla Firefox		Google Chrome
Version	**Shockwave Flash 17.0 r0**	**Shockwave Flash 18.0 r0**
Filename	**NPSWF32_17_0_0_188.dll**	**pepflashplayer.dll**

Type	Description	Type	Description
application/x-shockwave-flash	**Adobe Flash movie**	application/x-shockwave-flash	**Shockwave Flash**
application/futuresplash	FutureSplash movie	application/futuresplash	FutureSplash Player

Flash Plug-In Environment

	Mozilla Firefox	Google Chrome
Version	**WIN 17.0.0.188**	**WIN 18.0.0.160**
Player type	PlugIn	PlugIn
Manufacturer	**Adobe Windows**	**Google Pepper**
Operating system	Windows 8	Windows 8
Flash Architecture	x86	x86
Language	en	En
Screen Resolution	1600x900	1600x900
Screen Color	color	Color
Screen DPI	72	72
Pixel Aspect Ratio	1.0	1.0
Max Level IDC	5.1	5.1

Other system features

	Mozilla Firefox	*Google Chrome*
Flash Player with Debugger	× False	×False
AV Hardware Disable	× False	×False
Supports 32Bit Processes	✓ True	✓True
Supports 64Bit Processes	**✓ True**	**×False**
Has Accessibility	✓ True	✓True
Has Audio	✓ True	✓True
Has Audio Encoder	✓ True	✓True
Has Embedded Video	✓ True	✓True
Has IME	✓ True	✓True
Has MP3	✓ True	✓True
Has Printing	✓ True	✓True
Has Screen Broadcast	× False	×False
Has Screen Playback	× False	×False
Has Streaming Audio	✓ True	✓True
Has Streaming Video	✓ True	✓True
Has TLS	✓ True	✓True
Has Video Encoder	✓ True	✓True
Local File Read Disable	× False	×False
Windowless mode	× False	×False
Supports Dolby Digital Audio	? undefined	?undefined
Supports Dolby Digital Plus Audio	? undefined	?undefined
Supports DTS Audio	? undefined	?undefined
Supports DTS Express Audio	? undefined	?undefined
Supports DTS-HD Hi Res Audio	? undefined	?undefined
Supports DTS-HD Master Audio	? undefined	?undefined

Appendix IV – Example of OS and Java features, retrieved with Java Plugin

(Information retrieved from the website *BrowserLeaks.com* for the

Internet Explorer and *midori* browsers)

JVM properties	Internet Explorer	midori
Java version	1.8.0_45	1.8.0_45
Java vendor	Oracle Corporation	Oracle Corporation
Java vendor URL	http://java.oracle.com/	http://java.oracle.com/
JVM name	Java HotSpot(TM) Client VM	Java HotSpot(TM) Client VM
JVM version	25.45-b02	25.45-b02
JVM vendor	Oracle Corporation	Oracle Corporation
JVM specification version	1.8	1.8
JVM specification vendor	Oracle Corporation	Oracle Corporation
JVM specification name	Java Virtual Machine Specification	Java Virtual Machine Specification
Java specification version	1.8	1.8
Java specification vendor	Oracle Corporation	Oracle Corporation
Java specification name	Java Platform API Specification	Java Platform API Specification
Java class version	52.0	52.0
File separator	\	\
Path separator	;	;
Line separator	\r\n	\r\n
Java HTTP agent	Mozilla/4.0 (Windows 8 6.2)	Mozilla/4.0 (Windows 8 6.2)
JVM Uptime	**1999278**	**1871881**
JVM Start Time	**1435274207429**	**1435274336050**
Compiler Name	HotSpot Client Compiler	HotSpot Client Compiler
Total Compilation Time	**334**	**285**

Appendix V - Example of Network Interface Enumeration with Flash Plugin

(Information retrieved from the website *BrowserLeaks.com* for the

Internet Explorer browser)

VMware Virtual Ethernet Adapter for VMnet8						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth41	true	1500	false	true	false	false

VMware Virtual Ethernet Adapter for VMnet1						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth40	true	1500	false	true	false	false

Intel(R) WiFi Link 5100 AGN						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan0	true	1500	false	true	false	false

Software Loopback Interface 1						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
lo	true	-1	false	true	true	false

WAN Miniport (L2TP)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net0	false	-1	false	true	false	false

WAN Miniport (SSTP)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net1	false	-1	false	true	false	false

WAN Miniport (IKEv2)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net2	**false**	**-1**	**false**	**true**	**false**	**false**

WAN Miniport (PPTP)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**

net3	false	-1	false	true	false	false

WAN Miniport (PPPOE)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
ppp0	**false**	**-1**	**false**	**true**	**false**	**false**

WAN Miniport (IP)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth0	false	-1	false	true	false	false

WAN Miniport (IPv6)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth1	false	-1	false	true	false	false

WAN Miniport (Network Monitor)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth2	false	-1	false	true	false	false

Microsoft Kernel Debug Network Adapter						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth3	false	-1	false	true	false	false

RAS Async Adapter						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
ppp1	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection						
Interface	**Interface**	**Interface**	**Interface**	**Interface**	**Interface**	**Interface**
eth4	**eth4**	**eth4**	**eth4**	**eth4**	**eth4**	**eth4**

Intel(R) 82567LM Gigabit Network Connection						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth4	false	1500	false	true	false	false

Bluetooth Device (RFCOMM Protocol TDI)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net4	**false**	**-1**	**false**	**true**	**false**	**false**

Bluetooth Device (Personal Area Network)						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth5	**false**	**1500**	**false**	**true**	**false**	**false**

Microsoft ISATAP Adapter						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net5	**false**	**-1**	**false**	**true**	**false**	**false**

Microsoft Kernel Debug Network Adapter - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net6	**false**	**-1**	**false**	**true**	**false**	**false**

WAN Miniport (IP) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth6	**false**	**-1**	**false**	**true**	**false**	**false**

WAN Miniport (Network Monitor) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth7	**false**	**-1**	**false**	**true**	**false**	**false**

Intel(R) WiFi Link 5100 AGN - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth8	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth9	false	-1	false	true	false	false

Cisco Systems VPN Adapter for 64-bit Windows - Deterministic Network Enhancer Miniport

Interface	Is Up	MTU	Point-To-Point	Supports Multicast	Loopback	Virtual
eth10	false	-1	false	true	false	false

Microsoft ISATAP Adapter #2						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net7	false	1280	true	false	false	false

Microsoft 6to4 Adapter						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net8	false	-1	false	true	false	false

Teredo Tunneling Pseudo-Interface						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net9	false	1280	true	false	false	false

Microsoft Kernel Debug Network Adapter - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net10	false	-1	false	true	false	false

WAN Miniport (IP) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth11	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth12	false	-1	false	true	false	false

WAN Miniport (Network Monitor) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth13	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**

eth14	false	-1	false	true	false	false

Cisco Systems VPN Adapter for 64-bit Windows - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth15	false	-1	false	true	false	false

Microsoft ISATAP Adapter #3						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net11	false	-1	false	true	false	false

Microsoft Kernel Debug Network Adapter - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net12	false	-1	false	true	false	false

Cisco Systems VPN Adapter for 64-bit Windows - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net13	false	-1	false	true	false	false

WAN Miniport (IP) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth16	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth17	false	-1	false	true	false	false

WAN Miniport (Network Monitor) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth18	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth19	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN - Deterministic Network Enhancer Miniport

Interface	Is Up	MTU	Point-To-Point	Supports Multicast	Loopback	Virtual
eth19	false	-1	false	true	false	false

Microsoft Kernel Debug Network Adapter - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net14	false	-1	false	true	false	false

WAN Miniport (IP) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth20	false	-1	false	true	false	false

WAN Miniport (Network Monitor) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth21	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth22	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth23	false	-1	false	true	false	false

Cisco Systems VPN Adapter for 64-bit Windows - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth24	false	-1	false	true	false	false

Shrew Soft Virtual Adapter - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth25	false	-1	false	true	false	false

Microsoft ISATAP Adapter #4						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**

net15	false	1280	true	false	false	false

Microsoft ISATAP Adapter #5						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net16	false	1280	true	false	false	false

Microsoft ISATAP Adapter #6						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net17	false	-1	false	true	false	false

Microsoft ISATAP Adapter #7						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net18	false	-1	false	true	false	false

TAP-Windows Adapter V9 - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth26	false	-1	false	true	false	false

TAP-Windows Adapter V9 - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth27	false	-1	false	true	false	false

TAP-Windows Adapter V9						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth28	false	1500	false	true	false	false

TAP-Windows Adapter V9 - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth29	false	-1	false	true	false	false

GlobeTrotter HSxPA - Network Interface						
Interface	Is Up	MTU	Point-To-Point	Supports Multicast	Loopback	Virtual
eth30	false	-1	false	true	false	false

GlobeTrotter HSxPA - Network Interface - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth31	false	-1	false	true	false	false

Microsoft Windows Mobile Remote Adapter						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth32	false	-1	false	true	false	false

GlobeTrotter HSxPA - Network Interface - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net19	false	-1	false	true	false	false

Shrew Soft Virtual Adapter - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net20	false	-1	false	true	false	false

Microsoft Kernel Debug Network Adapter - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net21	false	-1	false	true	false	false

WAN Miniport (IP) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth33	false	-1	false	true	false	false

WAN Miniport (Network Monitor) - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth34	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth35	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth36	false	-1	false	true	false	false

TAP-Windows Adapter V9 - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth37	false	-1	false	true	false	false

Cisco Systems VPN Adapter for 64-bit Windows - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth38	false	-1	false	true	false	false

Microsoft Hosted Network Virtual Adapter						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan1	false	1500	false	true	false	false

Microsoft Hosted Network Virtual Adapter - Deterministic Network Enhancer Miniport						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth39	false	-1	false	true	false	false

Microsoft ISATAP Adapter #8						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net22	false	-1	false	true	false	false

Microsoft ISATAP Adapter #9						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net23	false	-1	false	true	false	false

Cisco Systems VPN Adapter for 64-bit Windows						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth42	false	-1	false	true	false	false

Shrew Soft Virtual Adapter						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth43	false	-1	false	true	false	false

Microsoft ISATAP Adapter #10						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net24	false	-1	false	true	false	false

Microsoft ISATAP Adapter #11						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
net25	false	-1	false	true	false	false

WAN Miniport (IP)-Deterministic Network Enhancer-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth44	false	-1	false	true	false	false

WAN Miniport (IP)-Shrew Soft Lightweight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth45	false	-1	false	true	false	false

WAN Miniport (IP)-QoS Packet Scheduler-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth46	false	-1	false	true	false	false

TAP-Windows Adapter V9-Deterministic Network Enhancer-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth47	false	-1	false	true	false	false

TAP-Windows Adapter V9-QoS Packet Scheduler-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth48	false	-1	false	true	false	false

TAP-Windows Adapter V9-WFP 802.3 MAC Layer LightWeight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth49	false	-1	false	true	false	false

TAP-Windows Adapter V9-Shrew Soft Lightweight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth50	false	-1	false	true	false	false

TAP-Windows Adapter V9-WFP Native MAC Layer LightWeight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth51	false	-1	false	true	false	false

WAN Miniport (IPv6)-Deterministic Network Enhancer-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth52	false	-1	false	true	false	false

WAN Miniport (IPv6)-Shrew Soft Lightweight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth53	false	-1	false	true	false	false

WAN Miniport (IPv6)-QoS Packet Scheduler-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth54	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN-QoS Packet Scheduler-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan2	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN-WFP 802.3 MAC Layer LightWeight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan3	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN-Virtual WiFi Filter Driver-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan4	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN-WFP Native MAC Layer LightWeight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan5	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection-Deterministic Network Enhancer-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth55	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection-QoS Packet Scheduler-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth56	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection-WFP 802.3 MAC Layer LightWeight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth57	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection-Shrew Soft Lightweight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth58	false	-1	false	true	false	false

Intel(R) 82567LM Gigabit Network Connection-WFP Native MAC Layer LightWeight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth59	false	-1	false	true	false	false

WAN Miniport (Network Monitor)-Shrew Soft Lightweight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth60	false	-1	false	true	false	false

WAN Miniport (Network Monitor)-QoS Packet Scheduler-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
eth61	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN-Native WiFi Filter Driver-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan6	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN-Shrew Soft Lightweight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan7	false	-1	false	true	false	false

Intel(R) WiFi Link 5100 AGN-Deterministic Network Enhancer-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan8	false	-1	false	true	false	false

Microsoft Hosted Network Virtual Adapter-Native WiFi Filter Driver-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan9	false	-1	false	true	false	false

Microsoft Hosted Network Virtual Adapter-Shrew Soft Lightweight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan10	false	-1	false	true	false	false

Microsoft Hosted Network Virtual Adapter-Deterministic Network Enhancer-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan11	false	-1	false	true	false	false

Microsoft Hosted Network Virtual Adapter-QoS Packet Scheduler-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan12	false	-1	false	true	false	false

Microsoft Hosted Network Virtual Adapter-WFP 802.3 MAC Layer LightWeight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan13	false	-1	false	true	false	false

Microsoft Hosted Network Virtual Adapter-WFP Native MAC Layer LightWeight Filter-0000						
Interface	**Is Up**	**MTU**	**Point-To-Point**	**Supports Multicast**	**Loopback**	**Virtual**
wlan14	false	-1	false	true	false	false