POLITECNICO DI MILANO
Computer Science and Engineering
School of Industrial and Information Engineering

Less is More *(secure)* Deprecator: automated reduction of the attack surface in modern Browser

Relatore: Prof. Stefano Zanero
Correlatore: Prof. Engin Kirda
Correlatore: Prof. Michele Carminati

Tesi di Laurea di:
Tommaso Innocenti, matricola 837744

Anno Accademico 2016-2017

Ringraziamenti

Finalmente sono alla fine di questo capitolo della mia vita, guardando indietro mi rendo conto di quanto io sia diverso e di quanta strada sia stata percorsa da quel primo giorno. Voglio ringraziare tutte le persone che hanno fatto parte di questo mio personale percorso, per averlo reso uno dei periodi più belli e pieno di ricordi di sempre.

Desidero ringraziare il prof. Stefano Zanero, per la gigantesca opportunità che mi ha offerto con questa tesi e per i tutti i suoi corsi che mi hanno introdotto al campo della sicurezza informatica. Ci tengo a ringraziarlo anche per il costante supporto che mi ha fornito durante tutta questa esperienza di tesi all'estero.

Un grande ringraziamento va al prof. Engin Kirda che mi ha accolto nel suo laboratorio a Boston e mi ha fin da subito fatto sentire parte del gruppo, ovviamente un grande ringraziamento va anche a tutto il gruppo del SecLab di Boston per il sostegno e i consigli.

Un ringraziamento speciale va al mio coinquilino Gino (Mamma africa) al quale devo dire grazie per essermi stato sempre vicino e avermi sempre sostenuto.

Un grazie speciale va a Giulia, la mia splendida ragazza, grazie per esserci sempre e per tutto l'aiuto e l'amore che mi hai sempre dato, spero riusciremo presto a vivere il nostro sogno americano.

Infine un grazie a tutta la mia famiglia per avermi sostenuto nei momenti difficili e per il grande supporto che mi avete sempre dato. Grazie per avermi aiutato a seguire i miei sogni e spronato a dare il meglio di me. Spero di essere in grado di ripagare tutti gli sforzi che avete fatto per sostenermi lontano da casa.

Contents

Abstract

Browsers, since their introduction, have started to integrate a large set of functionalities that are accessible by Web pages. Unfortunately this has attracted malicious actors that have started to use security breaches in all the parts of the browser to pursue their goals. Other than that, part of the available functionalities are barely used but, at the same time, they have sprung the growth of an invasive phenomenon: the *browser fingerprinting*. An invasive technique that, without the users' approval and without leaving any trace, allows Web sites to associate a unique ID to each user that visits the Web site. Monitoring this ID over the net allows to infer the Web history and the users' interests. The sensitive information obtained has been used to generate target advertisement able to increase the multi billionaire business of the big Internet players of the advertisement market.

In this thesis we present DEPRECATOR, an integrated module of Chromium, able to change the passive position of users, giving them the chance to decide which functionalities to expose to the Web sites. DEPRECATOR achieved this goal using an internal instrumentation of the functionalities of the browser. It was possible to generate a general model that encapsulates the usage of functionalities among Web sites, by using the functionality of DEPRECATOR to record the usage of the instrumented functionalities. The information extracted from the model identifies 25 standards that can be limited by the prevention system of DEPRECATOR over the 71 analyzed. Thanks to the internal approach chosen to integrate DEPRECATOR, our module is undetectable from the outside; this guarantees the impossibility for attackers or trackers to escape from the strict policy enforced.

DEPRECATOR has been tested over several measurements to ensure its performance and the goodness of the approach. This achieves a 28% of increase in the general performance compared with an unmodified browser. Our new module arose the privacy and security of the final user, showing it to be fully compatible with the first 100 Web sites of the Alexa ranking.

Sommario

Sin dalla loro prima comparsa, i browser hanno seguito un costante trend di crescita, aumentando considerevolmente le proprie funzionalità ed inevitabilmente incrementato la propria compressità interna. Tutte le funzionalità del browser sono rese disponibili ai siti internet, anche se la gran parte di queste ha mostrato uno scarso utilizzo. La politica di sicurezza delle Web API all'interno del browser è rimasta la stessa, anche se le funzionalità aggiunte in questi anni sono state numerevoli e hanno aperto nuovi scenari. Queste nuove funzionalità hanno fatto si che i browser diventassero la più grande piattaforma globale di sviluppo di applicazioni. Questo ruolo di prestigio unito alla quantità di informazioni sensibili gestite ha fatto si che attackers e malicious actors rivolgessero la propria attenzione verso i browser. Gli attacker hanno dimostrato di poter sfruttare le vulnerabilità in ogni parte del browser, ogni anno dimostrandosi in grado di superare le difese messe in atto dai browser. La buona notizia è che per superare i moderni sistemi di sicurezza è necessario la concatenazione di multiple falle di sicurezza, ne è l'esempio un recente exploit che concatena 14 vulnerabilità .

Oltre a questo la sempre più redditizia attività legata agli annunci pubblicitari su internet ha fatto si che le grandi aziende abbiano rivolto la propria attenzione verso i browser. Infatti vista la grande quantità di informazioni sensibili gestita, si è sin da subito capito che sfruttando tali informazioni si sarebbero poàtute creare campagne pubblicitarie mirate sui reali interessi degli utenti. Per ottenere tali informazioni le attività degli ignari utenti sono state registrate, si è così sempre più diffusa la pratica del tracciamento degli utenti sulla rete. In principio attraverso i cookies che sono stati largamente sfruttati in maniera impropria per fini di tracciamento degli utenti. Dopo che tale pratica è stata regolata e ritenuta illegale si è passati ad una più sofisticata tecnica di tracciamento degli utenti il *browser fingerprinting*, una tecnica che permette in maniera del tutto invisibile di tracciare gli utenti.

Il *browser fingerprinting* si basa sulla creazione di un Id unico per ogni utente che visita un sito internet, tracciandone il percorso sulla rete perme-

tte l'identificazione della cronologia e degli interessi dell'utente. La creazione di questo Id da parte dei siti internet si basa su sofisticate tecniche che permettono la raccolta di informazioni uniche relative al browser che visita il sito, tali da renderlo unico. La raccolta di tali informazioni è stata possibile grazie alla libertà di cui godono i siti internet, infatti i siti internet hanno accesso a molteplici funzionalità e sono in grado di sfruttarle a loro piacimento. Purtroppo gli utenti non hanno uno strumento per limitare questo invasivo fenomeno, essendo che non è possibile limitare le funzionalità rese disponibili. Oltretutto tale tecnica risulta invisibile da parte degli utenti, non modificando in alcun modo l'aspetto delle pagine, l'unica azione richiesta che ne consente l'attuazione è la visita da parte dell'utente della pagina internet.

La comunità scientifica ha rivolto la sua attenzione sulla questione della sicurezza dei browser e con lo studio di Nik Nikiforakis è stato evidenziato come questa tecnica del *browser fingerprint* può essere mitigata andando a manomettere i valori usati per identificare il browser tra gli altri browser. Lo studio condotto da Peter Snyder ha invece dimostrato come con lo sviluppo di un apposita estensione sia stato possibile limitare le funzionalità ben note per essere usate per tracciare gli utenti. Entrambi i lavori dimostrano come sia possibile mitigare questo fenomeno cosi invasivo, quello che però non riescono a fornire è una soluzione efficace che possa risolvere il problema. Infatti Nick Nikiforakis con il suo lavoro puramente dimostrativo afferma come sia possibile limitare la precisione di questo problema agendo sui valori usati. Mentre Peter Snyder dimostra come con un estensione sia possibile per un utente limitare queste funzionalità e avere un browser funzionante. La soluzione proposta è si di facile adozione per gli utenti, ma solo in rari casi le estensioni hanno ottenuto un'adozione tale da poter contrastare in maniera adeguata il problema affrontato.

Mossi dall'intenzione di restituire all'utente la possibilità di gestire e controllare il browser abbiamo sviluppato DEPRECATOR, un modulo interno di Chromium in grado di restituire all'utente il controllo sul browser e delle sue funzionalità. L'obbiettivo di DEPRECATOR è stato ottenuto grazie a un instrumentazione interna delle funzionalità del browser. L'instrumentazione del browser è stata automatizzata tramite l'uso di una apposita procedura che sfrutta gli IDL file presenti in Chromium facilitando l'individuazione delle funzionalità da instrumentare. L'instrumentazione delle funzionalità del browser ha permesso il monitoraggio di quest'ultime svolgendo un ampio studio di come queste funzionalità vengono usate dai siti internet. Da questo studio è stato possibile identificare 25 standard tra i 71 standard analizzati da rimuovere dal bowser. La rimozione di tali funzionalità è stata possibile

grazie all'uso della funzionalità di DEPRECATOR di prevenire l'esecuzione di alcune delle funzionalità instrumentate. Oltre a prevenire l'esecuzione di queste funzionalità il nostro modulo da all'utente la possibilità di specificare tramite una configurazione la propria policy di sicurezza: ovvero quale funzionalità esporre ai siti internet. La soluzione da noi proposta grazie all'approccio interno risulta invisibile dal punto di vista degli attackers o dei trackers, questo fa si che non sia possibile sfuggire alla rigida impostazione di sicurezza che impone il nostro modulo. Oltretutto la possibilità di integrare il sistema all'interno del browser suggerisce come sia possibile per i produttori integrare tale sistema nei browser, permettendo di elevare la sicurezza e la privacy di tutti gli utenti

Il modulo da noi proposto è stato in grado di offrire un aumento delle prestazioni generali del browser del 28%, tale valutazione è stata possibile grazie ai numerosi test ai quali è stato sottoposto DEPRECATOR. I risultati ottenuti oltre a mostrare un aumento di prestazioni sono stati in grado di mostrare la totale compatibilità del modulo con i primi 100 siti del ranking Alexa, e testarne la stabilità del sistema che si è dimostrata impeccabile durante lo svolgimento di tutte le misurazioni effettuate.

Chapter 1

Introduction

Internet has become part of everyone's daily life; most of people do not even realize how Internet has permeated their life. In every room of a house there is a device able to access the Internet: a smart scale, a smart TV, a surveillance IP-cam and other different devices. Even without considering these devices, it is evident that the society nowadays relies on the Internet infrastructure for almost everything: managing stocks, reading news, accessing home banking and even socializing.

The primary interface used from the users to connect to Internet and to perform the daily activities is the browser. Since its beginning as an hypermedia dissemination platform, browsers have become more and more important. In order to support all the different activities of all the users, the browsers have increased the available functionalities and have unavoidably increased also their internal complexity. This growth has been driven by the intense competition that browser vendors face against the popularity and the diffusion of native applications in smartphones. In order to compete, vendors have started to include in browsers a wide spectrum of capabilities like the access of the users' sensors, the webcam, the users' location and make asynchronous requests. These new introduced capabilities have transformed the web in the world's largest open application platform.

However, the security model underlying the browser Web API has remained largely unchanged. The security model applied grants to the Web pages the full access to the capabilities of the browser; mostly without having any chance for the users to limit them. The security model and the several sensitive information that pass by the browser have attracted year by year the interest of malicious actors. The attackers can leverage vulnerabilities in any part of the browsers' functionalities in order to exploit the user's machine and to be able to steal sensitive information or to install malware.

Browser vendors have tried to increase the security of the browsers, introducing multiple security mechanisms able to prevent or mitigate known vulnerabilities eg: the same-origin policy. This security mechanism consists in isolating each tab of the browser, dividing each tab in a different process and sandboxing each one of them. Despite these security defenses, every year browsers suffer from the discovery of new exploits, which had to take advantage of many vulnerabilities to successfully conclude the attack. A recent exploit [3] required a chain of 14 vulnerabilities in order to successfully exploit the browser.

Due to the relevance of the problem, and to the technical issue involved, the research community has moved its attention over browsers' security. One of the most relevant works is the analysis conducted by David Kohlbrenner et al. [18] that has demonstrated how it is possible to overcome the defense introduced by browser vendors and to continue accessing the possibilities to determine the exact precision of the clock. Accessing this functionality through the *timing channels* of a Web page's JavaScript code enables the execution of sophisticated fingerprinting code. Other relevant works [15, 19, 20, 8, 9] have studied how browsers can be used to track users among Web sites for advertisement purposes. The market size of the Internet advertisement [22] is remarkable; it has reached in 2017 223 billion dollars. Advertisement companies take advantage of all possible techniques to increase their profit using sophisticated methods and also browser's vulnerabilities to track users among domains to gather sensitive information.

As a consequence, researchers have developed several solutions. Nick Nikiforakis in his work [23]has shows that the *browser fingerprinting* issue can be mitigated by poisoning the known values used to track users. Also the work of Peter Snyder [27] has demonstrated, by developing an extension, how it is possible to restrict the access of some of the known features used to track the users. Both of the works agree with the idea that limiting the access of known features used to track the users will increase the security and privacy of the users. On the other hand, both of them lack in proposing an effective solution for the issue. The work of Nick Nikiforakis is a purely demonstrative work proving the goodness of the approach while the work of Peter Snyder proposes a solution that mitigates the issue but ignores the security implications related to the extensions and their real effectiveness. Indeed extensions are the first tool that allow users to integrate functionalities that are not part of the core of the browsers. The advantage of the extensions is the ease of develop it and the sophisticated actions that can be perform over the browsers. The problematic is that only in rare cases extensions have achieved a good rate of adoption able to mitigate the related issue

they face. Also several studies [11, 17] have warned about the criticality of vulnerabilities in extensions; this is due to the fact that browsers impose over the extensions a weak security policy. This privilege gives the ability of performing sophisticated actions to extensions but has also demonstrated how severe are the consequences of vulnerabilities in extensions. One other solution that has recently increased its popularity is the *Tor* browser; it tries to prevent the users' tracking while surfing the web. The solution adopted by *Tor* is trying to provide the same values for all the users to the known values used to track users; this mechanism, tries to provide the same fingerprint for all the users, reducing the precision of the fingerprint and ensures an higher level of privacy.

All these concerns moved the following question: is it a users' problem what browsers expose to Web pages? if it is so, how can users protect themselves against it? Surely users should be aware of how intrusive the tracking phenomenon is. For example, if one searches on the Internet for a new pair of shoes, a while after in every Web page visited appear shoes advertisements. To fight this invasive phenomenon is necessary an ethic discussion that involves not only experts that are trying to fight it, but also all the society. Nowadays, there is not a tool to completely prevent the browser fingerprinting, it is hard to distinguish the right usage of capabilities by Web sites and the malicious usage by fingerprinting scripts. What is demonstrated from the work of Peter Snyder [26] is that large part of the browsers' capabilities are not used from the majority of the Web sites. Attackers can leverage vulnerabilities in all parts of the browser, also the unused parts, this exposes the users to severe security implications.

The first question we want to answer with this thesis is: is it possible to exclude all these unused capabilities and still have a functional browser? The answer is positive, indeed DEPRECATOR is an integrated module of the Chromium browser able to restrict the usage of functionalities while preserving the functioning of the browser.

The design of DEPRECATOR draws on Peter's Snyder data set [26], indeed his data has been used as the starting point to the instrumentation of the browser's functionalities. To simplify the instrumentation of the Chromium browser we have designed an automated approach able to fully automatize the instrumentation process. The automated process uses *IDL file* to automatically find the target files where the functionalities of Chromium browser are implemented. Once the target files have been obtained, the browser is instrumented using a refactoring tool. This tool automatically inserts our module and compiles the code producing the modified version of the Chromium browser.

The characteristic that differentiates DEPRECATOR from the solution proposed by Peter Snyder in his work [27], is that our module uses and internal approach. The internal approach guarantees the invisibility from the outside of the browser. Attackers and trackers are not able to distinguish a browser with our module from one normal browser. This fundamental characteristic, combined with the hard-coding of the solution in the browser, prevents escaping from the strict security policy enforced.

Using the ability of our module to record the usage of instrumented functionalities, it has been possible to extend the analysis of the usage of functionalities among Web sites. The information obtained from this analysis allows the extraction and modeling the functionalities' behavior in Web sites. With these information it was possible to generate a configuration file used by DEPRECATOR that defines the functions that have to be restricted. The configuration file can be tailored over the needs of each user, the user can decide what to disable. Indeed the module allows the total configurabilities of the instrumented functionalities, this permits to limit the undesired functionalities. Thanks to this ability DEPRECATOR achieves the goal of giving back to the users the control over the browser and its functionalities. Also, in the default configuration file provided with the module, the known functionalities used to track the users are limited. With this configuration of the module, the collateral benefit is that the work of trackers becomes harder.

DEPRECATOR has been tested during several measurement in order to ensure its goodness, giving as result a 28% of increase in the general performance of the browser compared with an unmodified browser. DEPRECATOR has also demonstrated to be fully compatible with the first 100 sites of the Alexa ranking while improving the security of the entire browser.

The main contributions of this work are:

- The design of a mechanism able to automatize the instrumentation of the browsers
- A system able to give back to the user the control over the browser and its exposed functionalities
- The first integrated solution for the tracking issue

Chapter 2

Background and Motivation

In this chapter is first provided a brief introduction of the browsers' history, then is presented one of the most important threats that currently afflicts the browsers: the *browser fingerprinting*. Following are described the privacy implications of this invasive phenomenon and its evolution. Then, are described various solutions on this issue proposed by other related works. Finally is presented what we want to achieve with DEPRECATOR and the motivations that moved this work.

2.1 Browser evolution

Browsers provide the window to the Internet that million of users use daily. Technologies that support the interactions of users evolve with an incredible rapidity, lets just think about the computing power that twenty years ago was inaccessible and that now is in our hands. Browsers have followed the same path of evolution to embrace the new available technologies and new capabilities in order to adapt the users experience to the new opportunities available. This evolution has drastically changed how Internet is perceived from the users. At the beginning Internet was just a series of static pages, now Web pages embed all new kind of interactions and provide a dynamical interaction and a constant growing engagement for the users.

The Chrome browser, one of the most popular browsers, is a clear example of how fast a browser changes, it has a release cycle of just six weeks. This high frequency of changes is great: it allows a constant update of new standards and new capabilities although another side to the coin is that this constant addition of code has incredibly increased the code's complexity. Looking at the number of lines of the Chromium code [4] it is possible to see

that in 2016 it has overstepped the 20 MLOC[1] and on 2019 it is expected to reach the 30MLOC: For example, in comparison the latest[2] Linux kernel [5] has recently surpassed the 20MLOC. This incredible growth has been facilitated by the definition of standards, indeed almost all the functionalities provided by browsers are standardized by the W3C[3]. The standardization has permitted to have the same functionalities across multiple browsers increasing the compatibility but it has also shared the same problematic across all browsers.

Not all of the new capabilities are equally used, indeed most of the functionalities that have been added where due to the competition that browsers have to face with the native application of smartphones. Despite the fact that a large part of the browsers code has a poor usage, browser vendors are very wary in removing functions. Other than that browsers share the majority of the code base with the respective application for mobile devices, leaving on the desktop code functionalities that are designed for mobile devices. This means that JavaScript features that rely on hardware functionalities are included in the browser code without even addressing the problem if the machine has the hardware to support it.

2.2 Threats

Thanks to their increased popularity, to the quantity of sensitive information they processed and to the diffusion that they have achieved, browsers have attracted the attention of malicious actors. Malicious actors have found in browsers an ecosystem big as an operating system but with a weak security policy. Indeed browsers leave to the Web pages the full access to all the available capabilities, without providing the possibility to limit the exposed capabilities. Most of the users use the browser without even knowing all the possible security threats that they are facing. The largest part of the problems arises with the indiscriminate integration of functionalities within the browsers.

These additions have influenced negatively also the Web pages that have started to increase considerably their size, the increase is so big that nowadays it is tagged as the *Web Obesity Crisis* [13]. The crisis' effects are evident, one of these is the load time of Web pages that is increased due to the inclusion of heavy JavaScript codes and complex CSS that as result slow down the general experience of surfing the web.

[1] MLOC: Million of Lines of Code
[2] Version: 4.15.8
[3] W3C: World Wide Web Consortium

2.2.1 Browser fingerprinting

One of the most invasive problems that currently affects the modern browsers is the tracking of users across web domains. This invasive technique has been largely used for commercial purposes, indeed knowing the domains surfed by the users is a sensitive information that allows the creation of tailored strategies of advertisement that match the users' interests.

The Web pages that track the users, use sophisticated techniques to collect a rich fingerprint of the browser, these techniques allow the association of a unique identifier for each of the users that visit a site. This association enables the combination of multiple tracks of the same *unique id* across domains allowing the extraction of the history related to that *unique id*.

The Browser fingerprinting problem arouses around 2011 when the EU and the US law makers took actions against the Web sites to fight third-party cookies and to limit the unnecessary cookies to be used. Indeed until 2011 cookies have been largely misused from third parties to track users across domains and to retrieve their navigation history. From 2011 fingerprinting techniques have constantly improved their approach and precision taking advantage of all the possible data that can be extracted from the users' browser. The Web pages use JavaScript scripts to collect the fingerprint of the browser. Executing the fingerprinting code in the users' browser allows to collect a variety of information and to send it back to the servers. All the collected information is elaborated, permitting the extraction of *unique bits* of information that combined generate the *unique id* which uniquely identifies the user from all the others users.

Trackers decided to use the JavaScript code, that initially was included in the browsers to automatically update the GUI[4] of the browser, because its characteristic of dynamic code generation enables to load at run time libraries able to collect and generate a sophisticated fingerprint of the browser. This particularity of JavaScript allows third parties to escape all the static analysis of the code executed in a Web page. Besides this, combining multiple techniques such as *code obfuscation* and *dynamic code generation*, trackers can easily escape the weak restriction of Web pages.

Although there is to point out that the fingerprinting techniques have been also used to prevent malicious intents, indeed there are cases where they have been used to prevent fraud and cyberattacks. These cases are controversial with companies that use database of billions of devices id to determine the security risk associated with the devices used, claiming that fingerprinting is a "security" technique used to protect users. What is clear

[4]GUI: Graphical User Interface

is that a consciousness and a society debate are needed in order to regulate what is right and define what is bad.

Browser vendors on their side are largely unprepared to fight against this sophisticated phenomenon. They have tried to limit the precision of the fingerprint that can be extracted by adopting countermeasures that have revealed to be insufficient. Indeed the new sophisticated techniques used, combined with the adopted policy to expose every feature to the Web pages, makes ineffective all the recent efforts to fight this phenomenon.

2.3 Related work

The main goal of DEPRECATOR is not to solve the problem of browser fingerprinting but to give back to the final users the control of the browser letting the users decide what features expose to Web sites. One of the main implications of the goal accomplished by DEPRECATOR is to make harder the role of tracker limiting the exposed features that could be used to generate the fingerprint. In this section are presented the most relevant works about browser fingerprinting and the solutions proposed for this problem.

Firstly is presented the work of David Kohlbrenner et al. [18], this is one example that shows how the countermeasure adopted by browser vendors are weak and unable to limit the tracking phenomenon. Indeed David has demonstrated that it was possible to infer the exact precision of the clock even if the browser tried to provide by the *timing channels* a degraded precision of the clock to the JavaScript code. He has also discovered even other possibilities to infer the exact timing without using directly a *timing channel*. The *timing channels* are used to measure with a high precision the execution time of a defined portion of code, this information has been used from trackers to identify the device that has executed the code.

Every year there are new studies that discover new sophisticated techniques to fingerprint the defenseless users. One of the most recent is the work of Yinzhi Cao et al. [12] that has demonstrated how it is possible to generate a unique fingerprint of the user's machine even if the user uses different browsers to access a Web site.

This result has been enabled by the standardization of features across browsers, as previously mentioned the standardization has accelerated the growth and evolution of the features of browsers but has also shared the same problematic across browsers. Looking at the work of Yinzhi Cao it is clear how fingerprinting techniques have evolved their ability and have achieved a precision that allows the creation of the same fingerprint even across browsers. This means that third parties have achieved a precision

able to identify the machine that executes the browser. This information can allow an even more invasive track of people understanding if different accounts are managed on the same computer as it can usually happen in a family.

The evolution of fingerprinting techniques since 2010 is pretty impressive. One of the first studies of this phenomenon, the work of Peter Eckersley [14], using techniques that nowadays could be considered rudimentary, discovered the leak of 18.1 bits of *entropy* from browsers and he was still able to identify the 94.2% of the users starting the discussion on this phenomenon and the related implications.

2.3.1 Countermeasure proposed

Multiple researchers have tried to fight the phenomenon of tracking and purposed adequate countermeasure. Unfortunately protecting against this type of threat is difficult, the techniques used to collect sensitive information, looking from the users' point of view are invisible indeed to work, they do not need to store anything in the user's machine, the only needed action is to visit the Web page. Nowadays the users do not have a way to face completely against this abuse, indeed to prevent it the only solution will be to make inaccessible the values used to generate the fingerprint. Unfortunately it is hard to distinguish proper and malign usage and sadly the policy of browsers is to expose all the features available to the Web pages.

The work of Nick Nikiforakis et al. [9] proposes to poison the values captured by trackers with white lies randomizing the values collected to introduce a level of uncertainty in the fingerprint extracted. This approach tries to mitigate the phenomenon by showing that randomizing some values that are known to be collected by fingerprinting scripts would reduce the precision of the fingerprint collected. It is possible to argue that randomizing value could affect the behavior of benign sites or that collecting a non randomized value would reduce drastically the uncertainty created. Another problem that could be considered is the overhead introduced by altering the value or that the countermeasure is easy to be detected from the tracker, indeed requesting multiple times the poisoned value and comparing the values obtained, the fingerprinting script can detect it and adopt a different strategy.

A slight but similar approach has been used by the Tor browser [6] that, on purpose, tries to provide the same value across users to make it indistinguishable from one another. Other than that Tor will add a three layer of encryption over the system that manages the communication between user

and host, making impossible even for routers to link source and destination of data stream. The Tor browser, to achieve this level of anonymity, happily sacrifices the usability of the browser.

With the rapid growth and the new techniques adopted it is quite impossible to create a definitive solution that solves the problem. One solution for the problem is to push for a society debate on the problem that could stimulate the rethinking of the browser policy and improve the security and privacy of users. There is a positive example that is the ad-block extension that, in 2002, with the first release, started to fight the indiscriminate usage of advertisement, moved by a society discussion and desire it has moved the browser vendors to take actions directly. In 2018 [1]Google has included a functionality of ad-blocking in its browsers.

2.4 Motivation

In Section 2.1 has been presented the evolution of Browsers and in Section 2.2 one of the most important threats related to this indiscriminate growth of functionalities in the browser. Now it is presented what we want to achieve with this work: to prove that it is possible to change the direction of this growth trend. Starting from the fact that most of the functionalities are unused by the majority of the Web pages [26], we want to relieve the browser from all the heavy and potentially dangerous functionalities. DEPRECATOR, using its ability of limiting functionalities, will give back to the users the chance to control the browser. Using our module, the user can tailor the browser over its need, this implies having the desired functionality to perform the daily activities while limits as much as possible the undesired functionalities. The result of this customization is a browser able to perform the required tasks and at the same time able to provide a secure and fast experience to the user. The goals of the work could be summarized in:

1. To develop an automated tool able to transform any given version of Chromium to a *feature-instrumented browser*

2. To generate a map between high functionalities and low level features able, with an integrated mechanism, to reduce the attack surface and to give back to the user the direction over the browsers' functionalities

3. To develop the first integrated solution to the fingerprint issue that breaks the current state-of-the-art

Chapter 3

System Approach and Implementation

In this chapter is analyzed the fundamental design choice of the approach of the module we design. In the following section are illustrate two possible approaches, internal and external, highlighting their respective benefits and problematics. For both the approaches are provided examples from other related works. Then are illustrated the motivations that moved the choice of approach for DEPRECATOR, and how the challenges related to the chosen approach have been solved. Finally are described the design choice behind the chosen structure for our module and the process of integration in the Chromium browser.

3.1 Internal vs external

The choice of approach was between internal or external, as it can be imaginable each of the approaches has its benefits, and in this section both of them are presented with their own characteristics.

By external approach is meant a modification that does not change the code of the browser as it can be an extension. There are a lot of extensions able to arise the security of the browser, some examples could be *Ad-block* or *Ghostery*. The external modification of the browser has the benefit of being easy to develop and of still being able to perform sophisticated actions. Indeed the browsers for the extensions impose an even more permissive security policy compared to the normal Web pages. As it has been already presented in Section 2.2 the Web pages are under a more than enough permissive security policy. Thanks to this permissive policy, attackers have oriented their attention over the extensions, indeed extensions are allowed to perform ac-

tions in each tab of the browser and to extrapolate information from each tab. This permissive position allows the overcome of the security system implemented to isolate each tab and to impose the same-origin policy. There are multiple studies [29, 17, 11] that have discovered how extensions have been used to leak the private information of unaware users. Other than that extensions are visible from the outside of the browser, this implies that web sites could determine for the list of extension installed and it has been proven that trackers use this information to enrich the fingerprint of the browser.

The Chromium browser has modified the permission of installing extensions from unknown sources forcing a strict security policy that forces the developer of an extension to pass through a process of validation of the extension before it is available for the download in the extensions store. Even if Google has adopted this security check some extensions have been able to pass the check test and to perform malicious intent. To protect the users Google has designed a system that allows to register each installation of the extension over the browser and if an extension is reported to be malicious it can be remotely removed from the users' browser.

Looking at the browser fingerprinting issue the list of extensions installed could provide unique information used from trackers to enrich the fingerprint collected. Also recently Web pages have adopted a form of resistance against the extensions that block advertisements. Indeed some Web pages have started to limit the available articles that can be read by the users that have installed an ad-block extension. Other sites are trying to completely block the access to the site, forcing the user to insert in the *white list* the Web site or disallowing temporary the extension. Another different type of reaction by Web sites is the Web page of the *Boston globe* that is trying to restrict the access to the Web page for the users that use the incognito mode, forcing them to log in with an account in order to access the Web page and to read the articles with the incognito mode.

There are two factors that have contributed to the popularity of the extensions: the simplicity of integrating a solution in an extension and also their granted ability from the permissive policy of browsers. These two factors have made the extensions the ideal solution for a developer or a researcher who would want to develop a tool to protect against a new threat. An example is the work of Snyder et al. [27] that has chosen an extension to develop a system that intercepts JavaScript code by interposing a proxy object able to limit some undesirable functionality. The modification proposed by Snyder et al. has proved to be effective thanks to the large capability that an extension has in the browser. Concerning the compatibility the solution applied has achieved a breaking rate of the 15.71% during the test of the

first 200 sites of the Alexa ranking.

For the internal approach is intended the modification that changes the code of the browsers, as it can be the addition of a module or the instrumentation of a function. The internal modifications are harder to develop because, as explained in Section 2.1, the code of modern browsers could be compared in terms of complexity to the code of an operating system. Modifying directly the code allows to hard coding the modification making resilient the modification to adversaries as other extensions and other types of evasions from the modification imposed. Instrumenting the code allows to run the modification with the code of the browser making any modification of the code invisible from outside of the browser. A script or a Web site can not understand if the code of the browser executed is the original code or a modified code.

Researchers have used this approach primarily for testing and demonstrating purposes, one example is the work of Nick Nikiforakis et al. [23] that has modified the code of the Chromium browser to test the effectiveness of poisoning known values used to fingerprinting the browser. The work of Nick Nikiforakis has proven that poisoning the value is effective and that modifying the code directly affected less than 1% of the first 1,000 sites of the Alexa ranking. The internal modification has shown to be effective indeed the modify is deep and allows the modification to be direct and faster due to the fact that the modify is executed at the same level of the modified code. One big problematic that has the internal modification of the code is that it is time demanding and that, as explicated in the Section 2.1, the code of the browsers change fast and being able to provide the last version of the browser instrumented with the modification developed is a hard challenge that requires a big and constant effort.

The modifications of the browser's code are more effective in addressing the issue related with the browsers, even with never known, or the newest one that combines multiple mechanisms to evade the countermeasures. Indeed it has been recently discovered a new sophisticated technique of fingerprinting that uses an additional mechanism to hide its track, as the others it loads the JavaScript code at run time but after the execution of the script it will delete the part of the DOM where it has been executed. All of these attempts are ineffective against the internal modification indeed to accomplish their purpose the trackers need to run the code on the users' machine and an internal modification does not need sophisticated techniques to deobfuscate the code, it just needs to have the right countermeasure to adopt when malicious code is executed.

One example is the Tor browser that has been created starting from

a version of the Firefox browser that has been modified hard coding the modifications to the known features used to fingerprint the user, also to the modified browser has been added a layer of encryption to all the communications to ensure the anonymity of the users. Tor has shown how, from 2002, the desire of the users of surfing the web, while ensuring a higher level of privacy, has increased. This desire is reflected by the constant increase of the user base of Tor, in 2017 it was around three millions of users and it generated a total traffic over the net of 200Gbit/s [7].

3.1.1 Deprecator approach

For the development of DEPRECATOR it has been taken in consideration the internal approach. The motivation to choose the internal approach was driven by the desire to answer the following question: is it possible to eliminate the unused function and to still have a functional browser?

The answer is positive, indeed DEPRECATOR is an integrated module of Chromium able to reduce the attack surface of the browser. In the following Section 3.3 will be explained in details how it was possible to integrate it on top of the Chromium browser. One factor that has inclined to the choice of the internal approach was the higher compatibility that previous modifications have achieved compared with the external modification. Another important aspect is that the extensions have proven to be effective but they are lacking in terms of diffusion among users. Also, it is not desirable to have an extension that reduces the undesired functions but at the same time makes you more "unique" and so more fingerprintable.

To address the problem of reducing the attack surface the problem has been tackle from the inside to ensure if was possible to modify the permissive policy of the browser and still have a functional system. As anticipated in the previous section the internal approach introduces the challenge of providing continuously updates, indeed to modify the code of the browser is costly and with the high frequency, as the browsers are updated, it is an endless race.

To overcome this limitation it has been designed an automated procedure able to takes as input the functionalities that need to be instrumented and to give as output the instrumented version of the browser with DEPRECATOR linked to all the provided functionalities and ready to apply a strict security policy. The automated procedure was enabled by the usage of the *IDL file* that helped in the identification of the "target" inside of the code of Chromium and with the usage of the *clang compiler* was possible to automatically rewrite the code of the browser and to add DEPRECATOR to the code linking it to all the features to be checked. For more details in

the following Section 3.3.1 is explained how the process of instrumentation of the browser has been automated.

To reduce the attack surface and allow the customization of the browser, the starting point were the functions extracted from the work of Snyder [26]. The short coming of this work has been modified to create a map of the high functionality over the low level of features integrated in the browser. Then through the process of instrumentation was possible to insert the `hook`, a part of DEPRECATOR designed to intercept the execution of these features and to prevent the execution of the features that are defined as *potentially dangerous*. The detail of the system, able to intercept and prevent the executions of the instrumented features, is described in detail in the following Section 3.2.1.

What was possible to achieve with DEPRECATOR is the increasing of the overall security of the browser as it is demonstrated in the Section 4.1.3. It has been used the same internal approach of the Tor browser and of the work of Nick Nikiforakis, but with a solution that is a middle-of-the-road between the two works. Indeed the reduction of the attack surface includes some of the well known functionalities used to generate the fingerprint of browsers, this reduction would decrease the precision of the fingerprint generated and would provide a higher level of privacy to the users. Some of the functionalities related with the fingerprinting issue that have been limited are: the *WebGL*, *SVG vector*, *High resolution time*.

The solution proposed from DEPRECATOR is different, it bases its principles in the idea of giving back the control to the user that, with the high configurability of the solution proposed, could decide what to expose and what to limit. The high configurability of the solution adopted allows the generation of specific policies, this allows the user to expose the minimum set of necessary features for the Web sites and in this way reduces the breaking rate of Web sites. The solution proposed unaffected the usability of the browser while providing an increased secure browser with an exposed attack surface reduced. The system is provided with a *default configuration* that reduces the attack surface limiting 25 standards over the 71 analyzed, leaving totally unaltered the usability of the browser. For the full explication of how has been defined the default configuration the reader has to refer to the Section 4.1.2.

3.2 System Architecture

The structure of the module has been divided in two main components: "**the controller**" and the "**hook**". In the following sections will be explained the motivations behind each design choice that drove to the definition of each component and the interaction between them.

3.2.1 The hook

The first part of the system is the hook, this part is in charge to communicate with the controller and to apply the policy to the instrumented functions. In order to actively look, control and eventually prevent the execution of instructions is necessary to hook the current execution of these instructions and establish at run time the action to adopt in each case.

The hook is able, starting from a different context, to communicate with the controller giving it the **Function_id** of the instrumented function, this is the only information needed from the controller in order to reply with the respective action to undertake.

Instrumenting a copious number of functions required the design of a lightweight system that shall be capable of meeting the required functionality. With this objective in mind it was possible to reduce the implementation of this part of the system at just two lines of code.

As shown in the following listing a sample instrumented function:

```
1 ScriptPromise NavigatorBattery::getBattery(ScriptState* scriptState,
2                                             Navigator& navigator)
3 {
4    if(DeprecatorClass::GetInstance()->Block("Function_id")){
5    return ScriptPromise();
6    }
7   // function body
8   return NavigatorBattery::from(navigator).getBattery(scriptState);
9 }
```

Listing 3.1: Example of an instrumented feature

The second key aspect of the hook is the **prevention system**, indeed once the hook has received the policy to undertake from the controller, it needs to apply the relative action that could be: "*Allow*" or "*Block*" to the instrumented function.

DEPRECATOR's aim to reduce the attack surface is achieved by logically avoiding the execution of functions. Avoiding the normal execution of the function involves not only the instrumented function but also the **caller**, that is who has invoked the function and is waiting for a return value.

In order to preserve the integrity of the execution flow of Chromium the context of the called function must be considered. Returning a different type

of value could cause an unpredictable behavior of the system that could end in a segmentation fault. To preserve the integrity of the Browser in each instrumented function has been used a crafted hook with a return value able, when the normal execution of the function is avoided, to let continue the global flow without breaking it.

The choice of the returning value has been made using the IDL files where each function is defined as a guide to automate the extraction of the type for the return value. When it was not possible to automate it, a manual inspection of the code had to be done to find the right return value. In these few cases was necessary a manual inspection due to the relaxed definitions inside the IDL file. Indeed in the IDL file is permitted the definition of a function with two different return types.

In the code Listing: 3.1 the crafted return value is the *ScriptPromise()*, a constructor for the object *ScriptPromise* that generates an empty object: the *emptypromise*. This object enables the avoiding of that specific function using a safe object that preserves the integrity and stability of the browser.

3.2.2 The controller

The second part of the system is the controller, as previously explained one challenge that the controller needs to accomplish is to provide to each instrumented function the assigned action accordingly to the adopted policy. As well as sending a reply to each hook the controller is also able, using this communication channel, to track all the instrumented functions.

With the same instrumentation the system could be used for two purposes at the same time: to **enforce the policy** and to **collect data**. The instrumented functions spread all over the Chromium code need to refer to the same policy, in order to guarantee it the controller has been designed as a central control unit to store and manage policy. This design choice other than providing a central control reduces the overhead necessitating to load the policy to just one for each execution instead of loading it on demand. The management of the policy in one central point of the system is a key aspect that allows the controller to change it even at run time for all the functions at the same time, showing its effectiveness and flexibility.

Once the browser has been instrumented DEPRECATOR is able to change the behavior of instrumented functions without needing to recompile a single instruction but just changing the configuration file where the policy is defined. Also both the functionality of the system: **"collect data"** and **"enforce policy"** over the instrumented function are managed by the same configuration file achieving the same effectiveness and rapidity of action. The

configuration file provides a fine grain control that permits to change the behavior of each instrumented function. The behavior could be tailored over the needs of the user allowing to limit the exposed features based on the specific policy defined by the user.

DEPRECATOR will be provided with a **default configuration** with all the unnecessary features deactivated to simplify the approach of the inexpert user. For an extended overview of how it has been defined in the Section 4.1.2 is explicated the process of definition, also in the Appendix A is showed the extended list of standards restricted in the default configuration. The users do not need to become a security expert to use the system, once they familiarize with the security implications of activating features they could start managing the configuration and define their own policy. The configuration file also could be managed with an integrated service able to download the newest policy file and to apply it to the browser, as it commonly happens with the extension like ad-block, where the users automatically subscribe to a list where a community of experts define the list of blocked advertisement.

Acting in a very uncertain scenario where changes and new threats are discovered daily, software are forced to adapt as fast as possible to fix novel issues. Being able to change the behavior of an entire system changing just one file, even at run time, makes DEPRECATOR a desirable module to be included in every browser.

We could take as example the recent problematic that has afflicted the Battery API [24]. The research community has investigated and has discovered an `important security leak` due to the abuse of that API. Investigating further the community aroused a concern in the process of approval of a new API from the W3C. This discovery solicits the reply of the browser vendors that, in some cases with an unpredictable action vendors removed the API from the browser. A drastic approach that solves the problem but with a guiltily delay of months where the unaware user was exposed to this issue without having a tool to protect itself against it. Keeping this example in mind is clear how the design of browsers needed to be revised, introducing a faster security system able to increase the overall security.

Following is showed the functional schema of DEPRECATOR Figure: 3.1

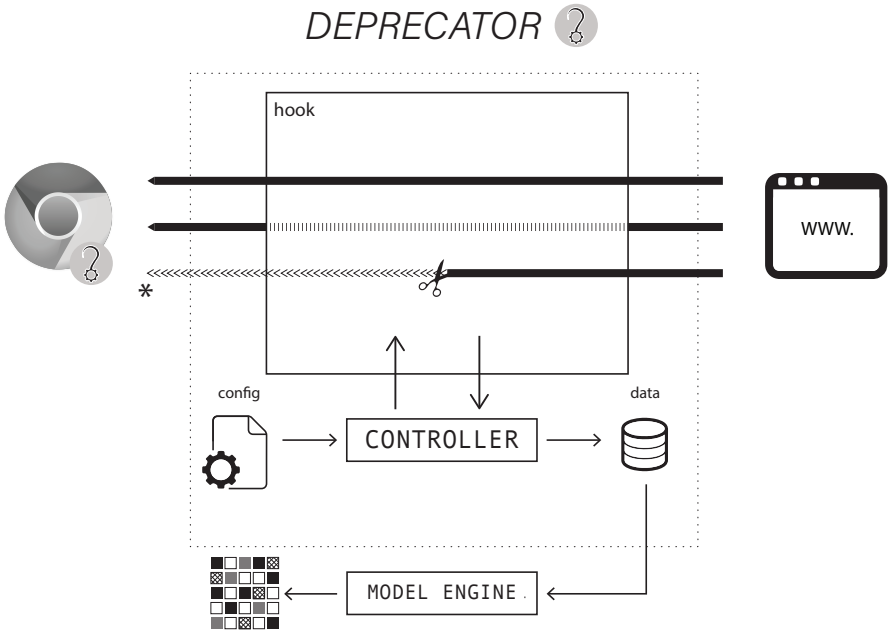


Figure 3.1: Functional Schema of DEPRECATOR.

The functional schema of DEPRECATOR Figure: 3.1 . The instrumented functions used from the website are treated in this way: to let go, to log, to use the prevention system (marked as *),to cut the normal flow of execution. The controller using the information extracted can enable the generation of the model based on these functions.

3.3 System integration

In this section is explained the integration process of the system in the Chromium Browser analyzing all the choices and solutions used to integrate it on top of the browser.

The integration process started with the controller choosing the *base folder* inside Chromium as destination. This folder contains all the classes and libraries that constitute the **basic blocks** used in Chromium. Before describing in details the process of adding the code of DEPRECATOR to Chromium is necessary a brief overview of the build system of Chromium.

Google has moved a process of migration more than two years ago from

GYP to GN as build system of Chromium, the new build system is twenty times faster then the previous one and more clear. The files that contain the instructions to build the code are more readable, making easy to understand the dependency between different modules. The *clang compiler* used to compile the code of the browser also offers tools that help the process of developing a new code. As example there is one tool able to check the correctness of dependency, a useful tool to prevent the creation of circular dependency.

Adding DEPRECATOR to Chromium's code implies the definition of the characteristics of the new module: the **structure** for the new code and the **dependency**, if there are. The structure of the code is chosen before modifying/creating the relative *BUILD.gn* file where are specified the characteristics of the new module. The compiler uses this file to link and compile the new module within the rest of the browser. There are different structures for the new code to choose among that are: *static library*, *shared library*, *source set* or *component*, all these structures vary the way of linking the code between modules.

The **component** structure has been chosen for DEPRECATOR. This structure is the recommended one for new modules, it makes the new code compile as a *shared library* if the build flag *is_component_build* is enabled or otherwise as a *static library*.
Once defined the **structure** it is necessary to generate a new build file or to modify an existing one adding on it the reference of the new code. For DEPRECATOR has been chosen to modify an existing *BUILD.gn* file, precisely the one in the *base folder*. The choice to modify this file in the folder *base* was made also for its visibility within the Chromium source code. As shown in the Scheme of dependency Figure: 3.2 the code in each rectangle can include only the code from the rectangles that are at the same level or under.

Placing the code in the *base folder* makes it respect the rule of dependency and makes it reachable from the instrumented code, that belongs to the *WebKit* part of the browser. Unfortunately this was not sufficient to allow the execution of DEPRECATOR because of **sandboxes**. Chromium has internal sandboxes that limit the available operations of the executed code within the browser [10].

The purpose of **sandbox** is to limit operations at process level that are classified as *possibly dangerous* if executed from an untrusted code. The sandboxed code in Chromium has these limitations: *opening new windows* and *accessing the disk*. *Opening new windows* from an untrusted code is classified as *possibly dangerous* letting the user surf on potentially malicious web

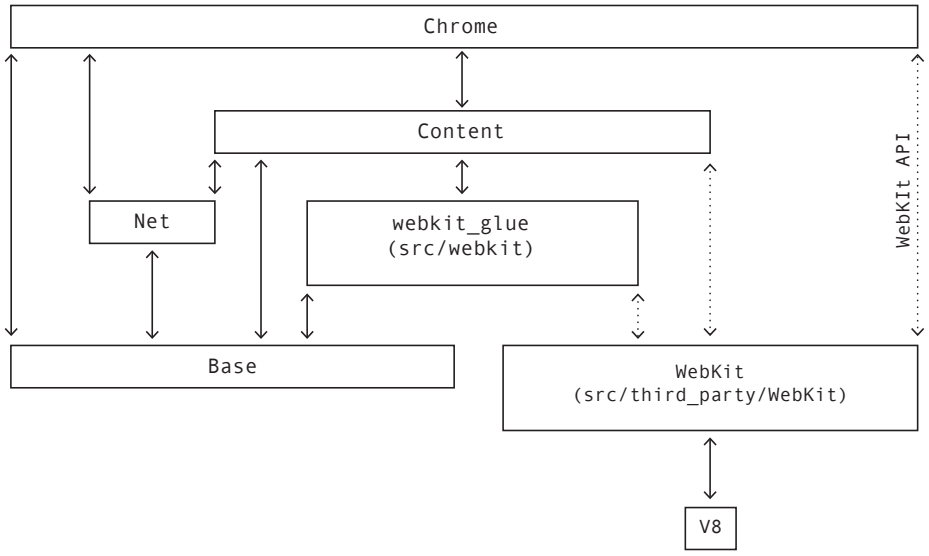


Figure 3.2: Diagram of Dependency in Chromium

pages without the user's approval. *Accessing the disk* is limited disallowing the possibility of stealing files or installing malware in the users' machine from untrusted process.

DEPRECATOR in order to work properly needs to "escape" from these strict requirements accessing the disk to load the configuration file. To overcome this limitation is necessary to run the Chromium Browser with the flag `--no-sandbox` active that disables the internal sandboxes. Due to the complexity of add our new module to an incredibly complicated system as Chromium, was necessary some compromises to operate properly.

The purpose of this work is to show the `feasibility` of a solution and to introduce the **first integrated solution** for a problem that every browser currently has. Hoping that this work will stimulate further investigations and works yielding to an integrated solution that raises the security of modern browsers and the awareness of the final users.

3.3.1 Instrumentation

After having introduced DEPRECATOR, its working principles and the relative interactions between system parts, in this section is explicated the process of instrumentation, starting from the preliminary data and finalization in the modified version of Chromium used to perform the measurement

and analysis.

The project is based on the study of the exposed attack surface in modern browsers especially measuring it as the **exposed features**. It draws on existing data sets: one of these is the work of Snyder et al. [26] that moved the motivation of inspecting deeper this aspect.
Choosing this work was a dutifully choice since it was the *most exhaustive measurement* of features in modern browsers focused on how these features are used among site instead of looking at all the threats spread on the net.

To perform all the measurements of this work the `Chromium browser` [1] has been used, this choice was driven by the popularity of Google's browser, in fact the only Chrome represents the 60% of the total diffusion of browsers [28]. A second but not secondary aspect is that Chromium and Chrome share the majority of the code's base, representing the closest example to the **most used browser** in the world that can be achieved using an **open source** browser.

Choosing Chromium dictated a preprocessing of the data sets that have been contemplated: considering [26], the first step was a conversion of the list of *features used* since they have used the Firefox Browser to perform their analysis. It was necessary to generate a list of *refined features* and to start a **matching process** of these functions in Chromium that ended in the positive match of more than **one thousand** single features. The features matched in the Chromium's code lead to the modification of a total of 184 file thought the procedure of instrumentation that is following described.

This preprocessing was necessary to exclude the proprietary functions of Firefox involved considering that such functions would not find any match in the Chromium browser. Also all the events have been excluded from the list of features to be matched, fundamentally for `two important factors` that are: first the decision of not considering it as part of the exposed surface of attack, indeed they are part of the system structure that needs it to work properly. Second the addition of events would end in an unsustainable volume of operations that have to be processed from our system that does not generate meaningful information. An event could trigger actions and if one of these actions requires a function that is considered part of the **exposed attack surface** this one will be inspected from DEPRECATOR, otherwise if it does not it will be fine to let it proceed without inspection.

[1] Version: 56,0,2920,0(64bit)

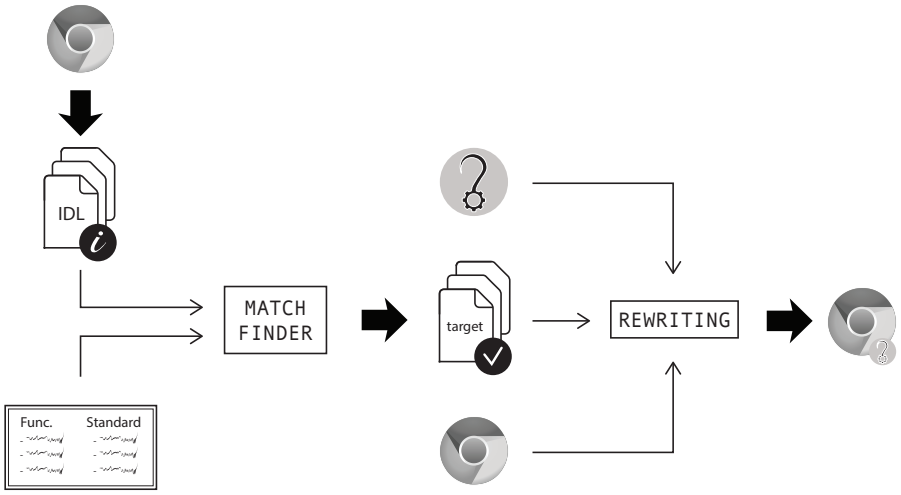


Figure 3.3: Instrumentation schema

As shown in the Figure 3.3, the instrumentation process is divided in two phases: the **match finder** and the **rewriting**.

The first part of the process is responsible of the **match functions** in Chromium, it makes use of the IDL [2] as a guide to matching features. The IDL is the standard language used to define the interface of WebAPI as a matter of fact it is the standard language used to define application programming interface. It has been chosen for its flexibility indeed it is **language-independent**, enabling the communication from two software components that do not even share a programming language.

When new APIs are released they come with documentation and IDL files that are provided as track to implement standard and features, browser vendors justly follow it to add these new standards and features in the browser. Even if there are cases where different browsers have used a slight different implementation [24]. We do not consider these corner cases that are beyond the scope of this work, taking Chromium as representative in terms of *implementation*.

As previously anticipated the **match finder** uses the *IDL definitions* to match features in Chromium. Now are provided more details of how IDL has been used in the matching phase and how IDL files are elaborated in Chromium clarifying their usage and purpose. IDL files are processed by the Chromium compiler to create the `JavaScript binding`. The binding

[2]IDL: Interface Definition Language

is C++ code, generated automatically, that is used by *V8 engine* to call *Blink*. This generation of code is divided in two main parts: "**front-end**" and "**back-end**".

The **front-end** part starts from the *IDL file* that is given as input for the **lexer** that generates the relative *tokens* as output, subsequently those token are transformed in an AST graph[3] by the **parser**, finally using the syntax tree it creates the **IR**[4].
In order to extract valuable information for the **matching finder** phase the considered part of this process is the **front-end**, because it is where the structure of the code, defined in the *IDL files* is processed to create the binding. The piece of software in charge of performing the first part of the code generation is the **idl_reader.py**, a module of the Chromium compiler wrote in *Python*.

The module has been modified to take as input also the list of functions to be matched, in this way was possible to `automate the matching` of these functions parsing all the *IDL files* of Chromium and finding the match inside the browser code. Other than finding the match for these functions, it was possible to associate `structural information` for each function as the return type. Later this additional information has been used to generate the **prevention system** and during the **rewriting phase** to select the proper **hook** to be used on each function.

The **matching phase** gives as result for each feature the exact file where the feature is implemented, this constitutes the **target**. Unfortunately this information is not enough to proceed with the `instrumentation` of Chromium. To proceed with the instrumentation is needed the file where is implemented the feature and also the exact location inside the file where to add the **hook**. This is the reason why of the second part of the instrumentation process.

For the **rewriting phase** has been used the *clang tool* to rewrite the code of Chromium and to insert the **controller** and the **hooks**. *Clang* provides the tool that supports the **global refactoring** of the Chromium code, this refactoring tool takes advantage of the **clang's AST**[5] to perform the refactoring. Indeed the *clang's AST* is more complete than other *AST* produced by different compilers making it suitable for refactoring operations.

The **rewriting phase** starts from a list of files where to search the implementation of the considered features. Each **target** of the list is analyzed by *clang* that generates the associated AST graph searching on it the

[3]AST: Abstract Syntax Tree
[4]IR: Intermediate representation
[5]AST: Abstract Syntax Tree

requested implementation and proceeds with the instrumentation. Once it has found the implementation of the feature in the file with the associated return value, that has been extracted from the previous IDL parsing, the right **hook** is selected to instrument the feature.

When the **rewriting phase** has concluded and so also the instrumentation process has ended the code of Chromium contains the changes ready to be compiled. Only the files where has been instrumented a function are recompiled making the instrumentation scale linearly with the numbers of files modified. All the actions of recompiling and linking on each modified file are automatically managed by the Chromium compiler that checks the related BUILD.gn file to ensure that all the rules are respected. At the end of all the control and once the instrumented code has been properly compiled the modified version of Chromium is ready to run taking advantage of the Deprecator module.

Chapter 4

Experimental validation

In this chapter are described the experiments performed on DEPRECATOR to assess the goodness of the approach and the implementation.

First is analyzed the **reduction of the attack surface** that the module was able to reduce, enforcing a security policy to the Chromium browser and the correlated security benefits of reducing the attack surface. The validation of the approach starts from the determination of where impose this reduction to the evaluation of the security benefits for the final user.

Finally is evaluated the **performance**, the **stability** and **usability** of DEPRECATOR, measuring its performance overhead in a real-world scenario, comparing it with a not instrumented version of Chromium. Due to its nature the browser is a critical system that needs to process sensitive information as quickly as possible. This characteristic dictates the design of the lightest as possible system. At the same time effortless for the user that could use a default setting and appreciate the improved security of the browser without needing to set complicated security policy.

4.1 Attack surface reduction

To determine where to apply the reduction of the exposed attack surface was necessary to understand how this considered surface behaves on different sites recording the stimulation of the instrumented functions. As previously explicated in Section 3.2.2 DEPRECATOR is able to collect data from the instrumented functions. This ability has permitted to create a map of low-level functions to a high-level of functionality simplifying its comprehension. With the collections of this low-level execution traces was possibly to identify the features used dividing at higher level the part of functionality that could be restricted without affecting the usability of the browser.

4.1.1 Experimental setup

The aim of the first measurement is to create an appropriate stimulation of browser instrumented with DEPRECATOR that could simulate the usage of the majority of the users. The stimulation has been made manually testing DEPRECATOR on the first 100 sites of the Alexa ranking for a total of 400 minutes of test. Using two different configurations and combining the data extracted has been generated a generalist model that encapsulates the features usage of both the configurations tested. Stimulating the sites means performing action like reading an article in a news site, watching a video, searching for an item in an e-commerce site, adding to the cart and starting a checkout process, download a software from a site, checking the fluctuation of the dollar, etc.

It has been decided to test DEPRECATOR on the first 100 sites of the Alexa ranking basing on the fact that the majority of the global web traffic is generated from these websites, taking as representative of the whole Internet. To ensure that the stimulation was effective, besides choosing the representative part of the net where testing the system, it has been decided how much time to spend in interacting with each site. Two minutes was the time necessary in order to stimulate adequately all the functionalities that the websites provides. Performing as much interaction as possible within the site for two minutes has been already proven [21] to be representative. With a dwell of more than a minute the stimulation encapsulated the normal scenario of an unauthenticated user that surfs on the net.

The two different configurations that were tested try to represent the majority of the users: the first one is a plain browser that the inexpert user uses it standalone to perform the daily activities on the net. Due to the goal of the test performed the browser used has been instrumented with DEPRECATOR to record the stimulation performed. The instrumentation is a transparent modification for the user, it does not change any aspect of the visual interface and without knowing that the browser used is instrumented, the user could not realize it. The second configuration tested is also this one an instrumented version of Chromium but with the addition of two popular extensions: *Ublock* and *Ghostery*. With the second configuration has been tried to embrace all the users that are more conscious of the privacy and security risks, users that have decided to use extensions that are able to mitigate threats that could bump into surfing the net. Nowadays for a common user the extensions are the only tool easily available to limit the possible threats of the net, without scarifying the usability of a common browser.

For both the measurements, the first with the plain browser and the second with the addition of the extensions, has been followed the same procedure of interaction described above. With these two measurements was possible to extract a model for each Website using the instrumented features used when the website is visited from an unauthenticated users. From the data extracted was possible to highlight different behaviors of Websites based on the configuration that requested the site. Indeed some functions were used only by one of the two tested configurations. As example the *Performance timeline* standard has been used only when the browser does not have any extension installed. The interesting behavior is that Websites stimulate differently the browsers' features when the user has extensions installed, even without showing different graphics and functionality. This result shows that Websites react differently based on the browsers' configurations that perform the request.

4.1.2 Data elaboration

The data extracted from DEPRECATOR on each site is the chronological usage of features during the stimulation of each website. The file size generated from DEPRECATOR varies among Websites from 3,8MB to 171MB, giving as result a total of 13 GB of files to be elaborated. In order to extract the information, the files are processed compressing the information. The compression of the information has been done by giving as input the files to a Python program the *model.py* that elaborates the extracted data in two phases. The first phase of the elaboration takes as input the 13GB of files to be elaborated, working site by site it will extract the following information:

- Name of the site
  - Standards used
  - Standards unused
  - Standards used only without extensions
  - Standards used only with extension
  - Number of invocation on each function without extension
  - Number of invocation on each function with extension

Using this information was possible to analyze the behavior of standards on each site understanding how standards have been used and their importance. Following in the Figure: 4.1 is showed the number of function calls associated with the standards used that has been recorded from DEPRECATOR during one of the test performed.

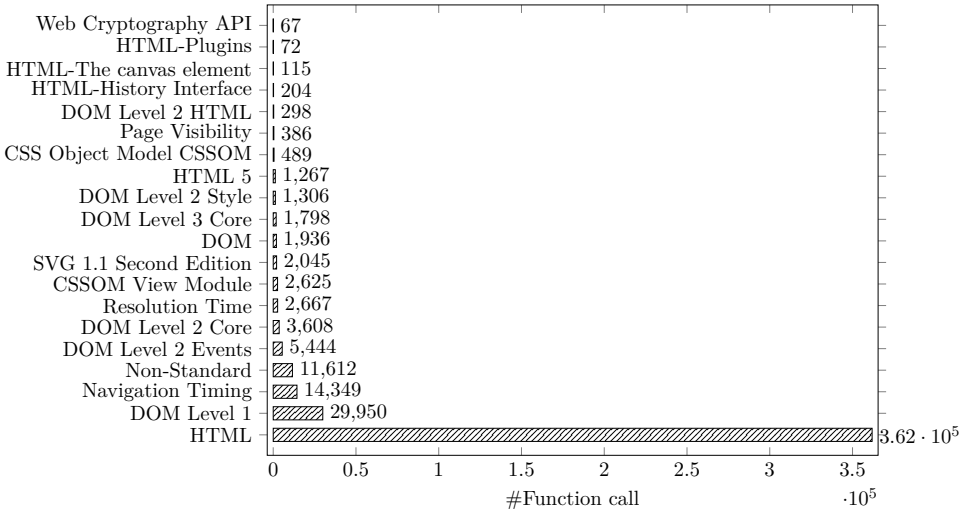


Figure 4.1: Number of function call on the first 20 most used standards

The schema shows the first twenty most used standards that have been measured. This particular schema of the standard usage is relative to *google.com* tested without extensions. On each of the measurements performed the data extracted has been analyzed to better understand the importance of the standards.

As it is clear from the schema, the standards with the higher number of function calls are the most used standards that has been recorded during the stimulation of the Web page. Analyzing the data extracted was clear that disabling the standard with the higher number of calls would most probably end in a error, making unreachable the Web site requested. Another analysis that could be extracted from this schema is that the majority of the functions calls are condensed only in a small portion of the standards.

These data show and confirm the hypothesis that, in order to deliver good service, the websites rely on original contents. The majority of the Web pages do not use the sophisticated interaction enabled by the multiple standards included in the browser code. This inactivity leaves a big portion of the code unused but available to malicious actors that are ready to use all the possibilities available to accomplish they malicious intention.

For the second phase the *model.py* now takes as input the information generated from the first phase, a file of 10MB, and generates a global model of behavior based on the instrumented functions considered. The model created is organized as follows:

- Standards always used
- Standards never unused
- Invocation range on each function without extensions
- Invocation range on each function with extensions

When the second phase of compression is completed the program gives as result a model of 161KB that encapsulated the general behavior observed from the two measurements. A compressed model more readable and understandable that enables the generation of a general policy for DEPRECATOR.

4.1.3 Result

Using the information extracted from DEPRECATOR and comparing with the previous result from [27] was possible to generate the model that has been used in the `default configuration`. The model generated disabled 25 standards over the 71 considered and their features that were considered as superfluous or *potentially dangerous* for the majority of the websites tested. For the extended overview of the standards restricted in the Appendix A is provided the complete list of standard restricted.

To express the security benefit gained has been used a statistical approach, indeed even well written code, as the Chromium code, is not excepted to contain defects/bugs, some software engineers estimate the defect density to be 3-6 per thousand lines of code [16, 25]. To have a measure of the benefit introduced, the number of lines of each instrumented function of the instrumented standard has been calculated by a manual inspection of the code.

In conclusion DEPRECATOR was able to exclude 5,419 lines of code with an estimation of 27 defects/bugs excluded from the browser. The obtained result is even more valuable if we consider that reducing the attack surface, reduces as well the odds of an attacker of exploiting a vulnerability successfully.

4.2 Performance overhead

The addiction of DEPRECATOR as an internal module of Chromium introduced an overhead in the execution of the browser due to the communication between its system parts. To quantify this overhead a series of test in a real scenario has been performed evaluating the performance of the instrumented browser against a not instrumented version of Chromium.

4.2.1 Page load time

To evaluate the impact of DEPRECATOR in a real scenario of usage has been considered the minimal daily activity performed within the browser that is, loading a Web page. The loading time of web pages is the minimal action repeated over and over during the day by the users. This action is able to show the effective capacity of the browser to be responsive supporting the interactions of the users.

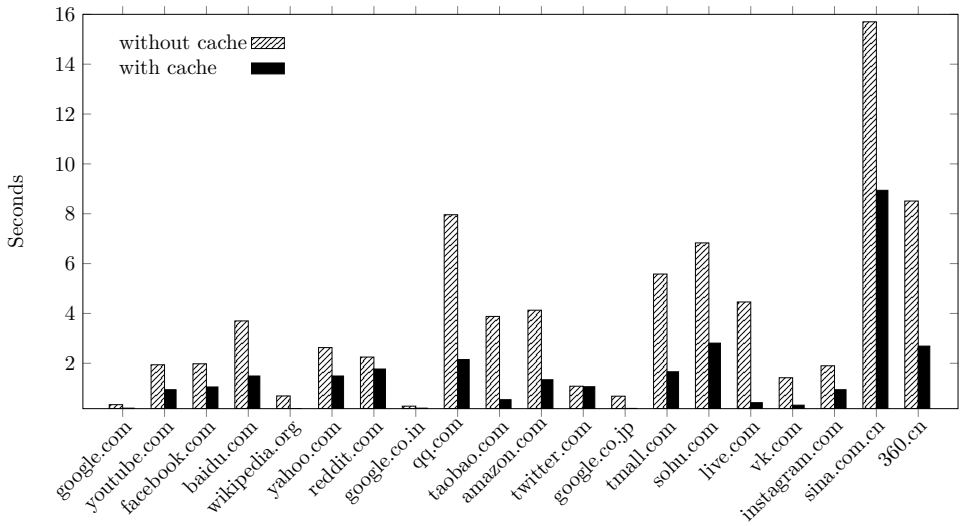
To measure the impact of DEPRECATOR in Chromium has been manually measured the load time of the first 100 Web pages according to the Alexa ranking. Each Web page has been firstly loaded by the Chromium browser instrumented with DEPRECATOR enforcing the security policy defined in the `Default configuration`. The second measurement has been performed using the unmodified version of Chromium to load the Web pages.

To measure the loading time of Web pages the *Page load time* [2] has been used, a popular extension used among Web developers to measure the performance of web page. The extension calculate the loading time of a Web page using the *high resolution time API*. The extension shows in each loaded page the time consumed in performing the necessary actions to load the page as: connect, request, response, load the DOM[1] and load the events. For the purpose of the tests has been considered only the total time necessitating to load the Web page.

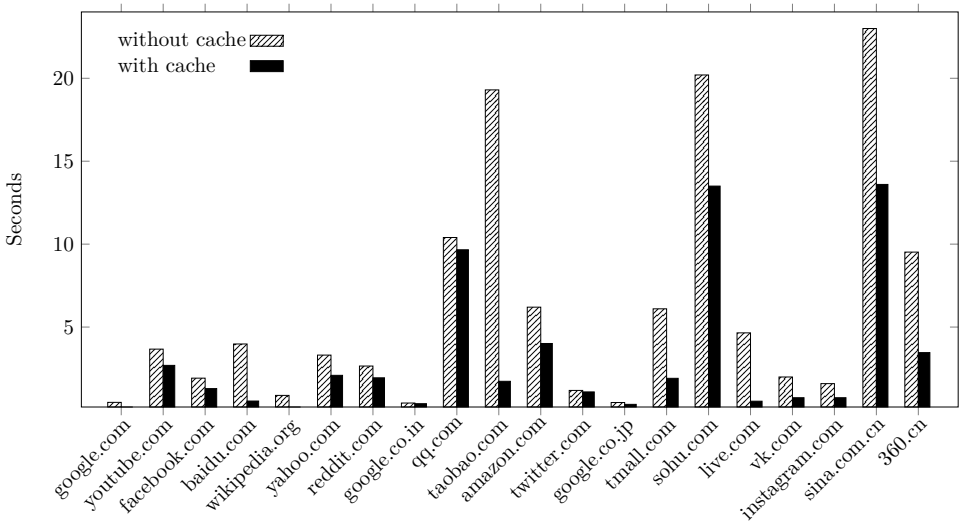
The measurements performed on each web page are two: one cold start and one warm load. The total measurements performed on each Web page are two on both the configurations, two with the instrumented browser and two with the normal browser, for a total of 400 pages loaded to compare the relative performance. Before each cold measurements, for both the configurations, the Chromium browser has been cleaned removing all the temporary data as cache, history and cookies. For the warm load after the first cold start of the page each Web page has been refreshed to evaluate the performance of the browser. In this way was possible to quantify the impact

[1] DOM: Document Object Model

of the cache in the page load and how the addition of DEPRECATOR could impact the refresh of pages. Following is showed the load time for the first 20 Web pages of the Alexa using the instrumented browser and the browser unmodified. A more extensive overview of the data extracted is provided in the Appendix B.



(a) Load time of Web page with DEPRECATOR



(b) Load time of Web page with Chromium

Figure 4.2: Comparison of the first 20 Web pages load time between DEPRECATOR *and Chromium*

As shown in the Figure 4.2 DEPRECATOR is able to arise the security of the browser, enforcing the *default policy*, and is still able to hide its execution overhead behind the performance advantage gained due to the code executions avoided. Indeed DEPRECATOR, using its prevention system, disables the functions that are classified as useless or potentially dangerous and reduces the numbers of lines executed from the browser. This code avoidance ends in a general improvement of performance, as it has been demonstrate from the test conducted.

In the following Table: 4.1 is reported the total time necessitating to load the first 100 sites of the Alexa ranking from each configuration. Also is quantified by the percentage the comparison of the two configuration tested, comparing the total time necessitating for DEPRECATOR to load the Web pages in both scenarios, with cache and without cache.

Browser configuration	DEPRECATOR		Chromium	
	Without cache	With cache	Without cache	With cache
Load time	304.72s	120.44s	404.91s	176.23s
Performance	75.25%	68.34%	100%	100%

Table 4.1: Total Performance comparison summary

DEPRECATOR has proven to outmatch the performance of the Chromium browser unmodified, this gain in performance is primary due to the avoidance of a big portion of code. Another factor that has positively contributed at this performance gain is the necessity of running Chromium with the flag *–no-sandbox*. As previously explained DEPRECATOR needs to escape from this restriction to work properly, but executing the browser in this modality allows the code executed within the browser to skip all the sandboxes restrictions. This is as it is warned from Chromium, at the startup of the browser with a warning message, a not secure setting that would affect performance and stability of the entire browser.

Trying not to look at this flagged message of warning, while running the browser with this setting, it will end up in an avoidance of part of the code of Chromium, introducing a relative performance improvement. This moved the curiosity of to inspect further if this intuition has a real foundation. To find proof of this hypothesis it has been decided to test the normal browser in the same way of the test previously performed. So it has been performed the measurement of the load time of the first 100 sites of the Alexa ranking with the unmodified Chromium browser executed with the flag *–no-sandbox* active.

Following is reported the Table 4.2 with the summary of the measurements performed:

Browser configuration	Chromium without sandbox		Chromium with sandbox	
	Without cache	With cache	Without cache	With cache
Load time	376.67s	243.01s	404.91s	176.23s
Performance	100%	100%	107.49%	72.51%

Table 4.2: Performance comparison of Chromium with sandbox and without sandbox

As partially intuited the avoidance of the internal sandbox gives back a little improvement in performance, more evident with the cold measurements. This improvement disappears during the warm measurement highlighting what was already clear before performing this measurement: to disable the sandbox is not a good idea. This further measurement was a meticulousness test to have an unbiased measurement of the performance gain of Deprecator. Considering the results of the test performed with the unmodified Chromium version it is possible to extract the final terms of benefit of using Deprecator that is around 28%.

4.2.2 Extraction of data

All the previous tests performed have measured the ability of Deprecator to enforce a strict security policy, that is its main purpose, but the system is also able to extract at run time information. In this section has been tested the performance of collecting data of Deprecator, it has been already proven the goodness of the data extracted but now has been tested the performance of running, enforcing the defined policy and collecting data at the same time.

As for the previous tests it has been used the same methodology, so the performance of the system has been tested loading the first 100 sites of the Alexa ranking and measuring the load time of Web pages. If in the configuration of Deprecator has been set as active the collection of data the system starts collecting data in real time from the startup of the browser. Indeed the trace of execution is updated continuously, this characteristic has been used to study the instrumented functions and to understand at run time, the correlation of the functions disabled and how the Web page reacts to the removal of these functions.

Following in the Table:4.3 is reported the summary result of all the tests performed:

Browser configuration	Deprecator		Deprecator + Log	
	Without cache	With cache	Without cache	With cache
Load time	304.72s	120.44s	803.22s	667.06s
Performance	100%	100%	263.59%	553,85%

Table 4.3: Performance comparison of the different modality of Deprecator

As shown from the table, enforcing a defined security policy and collecting data is a heavy task for the system, that introduces a significant overhead in the execution of the browser.

This modality has to be considered as a sort of debug mode of Deprecator, able to show what is going on and how the Web page stimulates the instrumented functions in real time. This modality has its advantage but enabling this modality will affect negatively the general performance. The normal usage of Deprecator is to run only enforcing the security policy. If a user is willing to understand what is going on with the instrumented functions the collection of the data could be modified in order to be less heavy considering the introduction of an interval of log. The system could receive the interval of log from the configuration file and update the trace of execution periodically instead of doing it in real time.

4.2.3 Usability and stability

The stability of Deprecator has been tested during the 600 tests performed to measure the performance and the identification of the attack surface reduction. During all the test even if the flag *–no-sandbox* was active Deprecator has exhibited an excellent stability and performance. All the Web pages tested were accessible and able to deliver its functionality, the only two exceptions are: *microsoftonline.com* and *googleusercontent.com*. These two Web pages were inaccessible due to the nature of the service that they provide. Indeed they are domains related to personal accounts that are used to access document loaded on the cloud, sync data and in general perform actions related to a specific account. These sites were inaccessible due to how the crawling has been performed. Indeed the choice of crawling the site without log in the sites visited was by purpose to simulate a scenario where the user has not any previous source of trust with the Web page. This situation is the perfect scenario where the users are prone to adopt a strict security policy before deciding that the site they are visiting could be

classified as a trust source.

The usability of DEPRECATOR is intuitive, once instrumented the browser and loaded the policy that enforces on the system, it does not require any further interaction by the users. The management of the configuration is the only part that requires the users' interaction. As previously anticipated, the management of the configuration could be automated with an automatic subscription to a repository, by setting a refresh rate of the configuration at the startup of the browser, making all the process effortless. The chance of changing the configuration at run time or with an automated service enables an easy development even for a big company, simplifying the diffusion and the change of the internal security policy.

Chapter 5

Limitation and future works

In this Chapter are provided the limitations of DEPRECATOR. Some of the limitations of our work are due to external factors while others to the nature of the issue treated.

5.1 Browser fingerprinting

Even with a large adoption of DEPRECATOR, the phenomenon of browser fingerprinting will continue to exist. Indeed as presented in the introduction, the market behind this phenomenon is large, the techniques adopted have shown a fast growth in terms of precision and there will still be parts of the browser or an unintended composition of features that will be able to give to the trackers sensitive information about the users.

What is reasonable to expect is that with a large adoption of DEPRECATOR the majority of the users will configure a restrictive policy with the consequence of a decrease in the precision of this invasive method. The combination of what has been previously said with a general awareness of the phenomenon would allow the production of a dense fog between our private data and the trackers, making harder the role of trackers.

5.2 Limit of the approach

Even if we took in consideration a large set of features to be analyzed, not all the parts of the browser where covered by this measurement.

The automatization of the instrumentation process is afflicted by one of the limitations, indeed part of the process is based on the usage of *IDL files*, unfortunately, as explained in Section 3.2.1, these files were not designed for this purpose In a few cases, the choice of using these files made necessary a

manual inspection of the code in order to fix the automated instrumentation and to choose the right hook to use.

Although the overall performance and precision of the instrumentation process was able to indicate that the approach was heading toward the right direction, indeed the cases of manual inspection were limited and a better inclusion of these corner cases will provide a full coverage of them. With the used approach it is possible to extend the coverage to new features and new standards with a limited amount of work. Also, the extension of this approach to other browsers is possible with a restricted amount of work, adding other browsers to the compatible system of DEPRECATOR.

5.3 Future work

In the future development of DEPRECATOR one of the most relevant aspects is the adoption of a more sophisticated system able to determine the origin of a function call. Understanding the context would enable the chance to determine first party or third party inside the code executed from the browser and to limit only the desired part. This could allow the definition of specific policies that work site by site, or even *iframe* by *iframe* to guarantee the best experience to the users among every site.

Another improvement of easy adoption could be the possibility to temporarily disable the enforced policy, indeed for the users could be necessary to disallow a specific policy to access to a system that needs some restricted features, this process could be automated by an automatic function that would reset the policy enforced with just one click. This additional functionality takes advantage of the high configurability of DEPRECATOR, indeed it can change the behavior of instrumented functions even at run time.

One desirable future scenario would be the definition of specific security profiles of Web pages, defined by Web developers and checked by external authorities. These profiles could be used to show the level of agreement within the visited site, comparing the security profile defined by the user and the public security profile of Web pages. These definitions could be used to understand drifting behavior of well known Web pages using the security profile where are declared the functionalities necessary from the site to deliver its functionalities. In this ideal world would start a competition among Web developers to use the minimum set of features and to deliver the most engagement user experience possible.

Chapter 6

Conclusions

We presented DEPRECATOR, a new module of the Chromium browser able to customize the browser, that allows the users to customize their own configuration. With the personalization of the configuration the user can determine which functionalities of the browser to use and what to limit.

Taking advantage of the abilities of our module to record the usage of the instrumented functionalities we performed a large study of the browsers' functionalities. The study performed tries to embrace the usage made by the majority of the users while surfing the web. Using the data collected it was possible to identify and model the usage of the browsers' functionalities. The obtained information from this model has been used to define the *default configuration*, where the functionalities of 25 standards over the 71standards considered have been limited.

To test the goodness of the model we measured the default security policy on the first 100 sites of the Alexa ranking. The result showed the fully compatibility with the sites tested. Even with the restricted set of functionalities, all the sites were functional and able to provide its functionality. Moreover the *default configuration* we provided with the module is only one example of configuration that can be used with DEPRECATOR. The high possibility of customizing the configuration allows the definition of even strict security policies. The management of the configuration file can be both manual, for the expert users or fully automated to help the inexpert ones.

Our module goes in the reverse trend by removing functionalities, indeed it focuses in providing just the necessary set of functionalities to the final user that is free to appreciate a lightweight and also faster browser. Indeed, from the result of the tests performed, the avoidance of parts of the code, produces an improvement in the performance of 28% compared with the unmodified browser. Taking advantage of the designed instrumentation it is

possible to provide updates of our module and to integrate it in the future versions of the Chromium browser.

Thanks to the internal approach our module is undetectable from the attacker's point of view. Also, with the limitation of the exposed functionalities, the browser is able to provide a more secure experience to the users. With the default configuration we were able to achieve the collateral benefit of making harder the work of trackers. Indeed in the functionalities limited by the *default configuration* have also been included some of the functionalities used to track the users.

we so hope with our work to stimulate further investigation that could possibly produce an integrated solution that could be added in all the modern browsers.

Bibliography

[1] Chrome ad filtering. https://blog.chromium.org/2018/02/how-chromes-ad-filtering-works.html.

[2] chrome-load-timer. https://github.com/alex-vv/chrome-load-timer.

[3] Chromium blog. A tale of two Pwnies. https://blog.chromium.org/2012/06/tale-of-two-pwnies-part-2.html.

[4] Chromium code statistics. https://www.openhub.net/p/chrome/analyses/latest/languages_summary.

[5] Linux code statistics. https://www.linuxcounter.net/statistics/kernel.

[6] Tor browser design. https://www.torproject.org/projects/torbrowser/design/#fingerprinting-linkability.

[7] Tor browser metrics. https://metrics.torproject.org/bandwidth.html.

[8] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, pages 674–689, New York, NY, USA, 2014. ACM.

[9] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pages 1129–1140. ACM, 2013.

[10] Adam Barth, Collin Jackson, Charles Reis, TGC Team, et al. The security architecture of the chromium browser. Technical report, 2008.

[11] Ahmet Salih Buyukkayhan, Kaan Onarlioglu, William K Robertson, and Engin Kirda. Crossfire: An analysis of firefox extension-reuse vulnerabilities. In NDSS, 2016.

[12] Yinzhi Cao, Song Li, and Erik Wijmans. browser fingerprinting via os and hardware level features. In Proceedings of Network & Distributed System Security Symposium (NDSS), 2017.

[13] Maciej Ceg?owski. The Website Obesity Crisis. `http://idlewords.com/talks/website_obesity.htm#crisis`.

[14] Peter Eckersley. How unique is your web browser? In International Symposium on Privacy Enhancing Technologies Symposium, pages 1–18. Springer, 2010.

[15] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pages 1388–1401, New York, NY, USA, 2016. ACM.

[16] Les Hatton. Reexamining the fault density-component size connection. IEEE Softw., 14(2):89–97, March 1997.

[17] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In USENIX Security Symposium, pages 641–654. San Diego, CA, 2014.

[18] David Kohlbrenner and Hovav Shacham. Trusted browsers for uncertain times. In USENIX Security Symposium, pages 463–480, 2016.

[19] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In Security and Privacy (SP), 2016 IEEE Symposium on, pages 878–894. IEEE, 2016.

[20] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In USENIX Security Symposium, 2016.

[21] Chao Liu, Ryen W. White, and Susan Dumais. Understanding web browsing behaviors through weibull analysis of dwell time. In Proceedings of the 33rd International ACM SIGIR Conference on

Research and Development in Information Retrieval, SIGIR '10, pages 379–386, New York, NY, USA, 2010. ACM.

[22] Rani molla. Global net ad revenue share for digital and mobile 2017. https://www.recode.net/2017/7/24/16020330/google-digital-mobile-ad-revenue-world-leader-facebook-growth.

[23] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies. In Proceedings of the 24th International Conference on World Wide Web, pages 820–830. ACM, 2015.

[24] Lukasz Olejnik, Steven Englehardt, and Arvind Narayanan. Battery status not included: Assessing privacy in web standards.

[25] Andy Ozment and Stuart E Schechter. Milk or wine: does software security improve with age? In USENIX Security Symposium, pages 93–104, 2006.

[26] Peter Snyder, Lara Ansari, Cynthia Taylor, and Chris Kanich. Browser feature usage on the modern web. In Proceedings of the 2016 ACM on Internet Measurement Conference, pages 97–110. ACM, 2016.

[27] Peter Snyder, Cynthia Taylor, and Chris Kanich. Most websites don't need to vibrate: A cost-benefit approach to improving browser security. CoRR, abs/1708.08510, 2017.

[28] World Wide Web Consortium (W3C). Web browser usage trend.

[29] Michael Weissbacher, Enrico Mariconti, Guillermo Suarez-Tangil, Gianluca Stringhini, William Robertson, and Engin Kirda. Ex-ray: Detection of history-leaking browser extensions. In Proceedings of the 33rd Annual Computer Security Applications Conference, pages 590–602. ACM, 2017.

Appendix A

Default configuration

In this chapter are showed the standards that have been included in the `Default configuration`, the configuration ensures a strict policy that has proven to arise the security of the system and the privacy of the user while being fully compatible with the 100 sites of the Alexa ranking tested. If compared with the policies applied by the normal browser it is a strict policy but nevertheless the user is free to change it and adopt an even more restrictive policy, balancing the trade of between security and usability.

Default configuration	
Standards	Restricted
WebVTT:The Web Video Text Tracks Format	✓
Beacon	✓
WebGL Specification	✓
Web Audio API	✓
Vibration API	✓
Media Capture from DOM Elements	✓
Console API	✓
UI Events Specification	✓
DOM Parsing and Serialization	✓
DeviceOrientation Event Specification	✓
Shadow DOM	✓
Performance Timeline Level 2	✓
Pointer Lock	✓
Resource Timing	✓
Scalable Vector Graphics (SVG) 1.1 (Second Edition)	✓
Selection API	✓
MediaStream Recording	✓
Indexed Database API	✓

Default configuration	
Standards	Restricted
High Resolution Time Level 2	✓
HTML-Channel Messaging	✓
HTML-Web Sockets	✓
Gamepad	✓
Fullscreen API	✓
HTML-Web Workers	✓
Encrypted Media Extensions	✓
Fetch	X
File API	X
Geolocation API Specification	X
Geometry Interfaces Module Level 1	X
HTML	X
Encoding	X
HTML-Broadcasting to other browsing contexts	X
HTML-The canvas element	X
HTML-Plugins	X
HTML-Web Storage	X
HTML-History Interface	X
HTML 5	X
HTML 5.1	X
Document Object Model (DOM) Level 3 XPath Specification	X
Document Object Model (DOM) Level 3 Core Specification	X
Media Capture and Streams	X
CSSOM View Module	X
Media Source Extensions	X
Document Object Model (DOM) Level 2 Traversal and Range Specification	X
Navigation Timing	X
Non-Standard	X
Page Visibility (Second Edition)	X
Performance Timeline	X
Document Object Model (DOM) Level 1 Specification	X
Document Object Model (DOM) Level 2 Core Specification	X
Document Object Model (DOM) Level 2 Events Specification	X
Document Object Model (DOM) Level 2 HTML Specification	X
Document Object Model (DOM) Level 2 Style Specification	X
Selectors API Level 1	X
Service Workers	X
The Screen Orientation API	X
Timing control for script-based animations	X
DOM	X
Tracking Preference Expression (DNT)	X
URL	X

Default configuration	
Standards	Restricted
User Timing Level 2	X
W3C DOM4	X
Battery Status API	X
CSS Font Loading Module Level 3	x
CSS Object Model (CSSOM)	X
Web Cryptography API	X
Web Notifications	X
CSS Conditional Rules Module Level 3	X
WebRTC 1.0:Real-time Communication Between Browser	X
XMLHttpRequest	X
execCommand	X

Table A.1: Standard included in the Default configuration

Appendix B

Full data performance evaluation

In this chapter are reported the full data collected during the test to measure the performance of DEPRECATOR. In the first column of the table are reported the load time of the first 100 sites of the Alexa ranking loaded with the browser instrumented with DEPRECATOR, the second column of the table contains the load time measured with the browser instrumented with DEPRECATOR that other than enforcing a strict policy will also collect the data of the instrumented features in real time. In the second table are reported the load time performed with the normal browser, in the first column the load time of the browser executed with the sandbox and in the second column without the sandbox. The measured load time of each page, for all the measurements, has been extracted using the extension page load time.

Web site	Browser configuration			
	DEPRECATOR		DEPRECATOR + Log	
	Without cache	With cache	Without cache	With cache
1)google.com	0.34s	0.20s	1.09s	0.91s
2)youtube.com	1.94s	0.94s	13.6s	11.1s
3)facebook.com	1.98s	1.05s	2.42s	1.70s
4)baidu.com	3.70s	1.49s	5.64s	1.99s
5)wikipedia.org	0.69s	0.18s	3.01s	2.58s
6)yahoo.com	2.63s	1.49s	25.7s	25.9s
7)reddit.com	2.25s	1.77s	16.8s	7.51s
8)google.co.in	0.28s	0.20s	1.37s	0.96s
9)qq.com	7.96s	2.15s	20.1s	10.2s
10)taobao.com	3.88s	0.54s	5.56s	1.66s
11)amazon.com	4.13s	1.34s	9.45s	8.51s

Web site	Browser configuration			
	Deprecator		Deprecator + Log	
	Without cache	With cache	Without cache	With cache
12)twitter.com	1.08s	1.06s	2.80s	2.95s
13)google.co.jp	0.68s	0.19s	1.27s	0.99s
14)tmall.com	5.58s	1.66s	6.68s	4.07s
15)sohu.com	6.83s	2.81s	34.6s	35.1s
16)live.com	4.46s	0.42s	4.51s	1.35s
17)vk.com	1.42s	0.32s	4.90s	3.04s
18)instagram.com	1.90s	0.94s	4.63s	4.03s
19)sina.com.cn	15.7s	8.94s	63.2s	58.1s
20)360.cn	8.51s	2.69s	9.86s	5.31s
21)jd.com	4.95s	0.16s	5.57s	1.54s
22)google.de	0.63s	0.18s	1.25s	0.95s
23)google.co.uk	0.63s	0.19s	1.25s	0.95s
24)linkedin.com	1.33s	0.59s	6.95s	5.92s
25)weibo.com	9.92s	2.97s	17.0s	7.15s
26)google.fr	0.65s	0.18s	1.16s	0.94s
27)google.ru	0.68s	0.19s	1.40s	1.02s
28)google.com.br	0.57s	0.18s	1.22s	0.98s
29)yandex.ru	2.57s	1.06s	7.48s	4.36s
30)yahoo.co.jp	11.3s	6.72s	12.3s	11.8s
31)netflix.com	1.46s	0.90s	2.38s	1.73s
32)google.com.hk	0.66s	0.19s	1.28s	0.98s
33)t.co	0.35s	0.19s	0.52s	0.34s
34)hao123.com	8.94s	3.99s	14.6s	12.2s
35)imgur.com	3.53s	1.48s	8.60s	5.61s
36)google.it	0.28s	0.18s	1.07s	0.92s
37)ebay.com	2.48s	1.39s	7.02s	5.46s
38)pornhub.com	2.72s	1.77s	8.77s	7.38s
39)google.es	0.59s	0.26s	1.24s	0.97s
40)detail.tmall.com	3.07s	1.03s	4.02s	2.15s
41)wordpress.com	2.09s	1.29s	7.39s	5.97s
42)msn.com	5.20s	1.22s	13.3s	9.81s
43)aliexpress.com	3.20s	0.83s	9.36s	5.65s
44)bing.com	0.42s	0.62s	1.27s	0.97s
45)tumblr.com	1.98s	0.71s	10.6s	6.31s
46)google.ca	0.63s	0.18s	1.26s	0.96s
47)microsoft.com	0.81s	0.38s	3.79s	3.98s
48)livejasmin.com	3.27s	1.65s	12.4s	10.2s
49)stackoverflow.com	2.04s	0.73s	7.67s	6.73s
50)twitch.tv	2.05s	0.44s	3.87s	3.68s
51)ok.ru	2.29s	0.72s	6.39s	4.67s
52)google.com.mx	0.65s	0.20s	1.24s	0.96s
53)ntd.tv	6.98s	1.87s	22.2s	17.6s
54)onclkds.com	0.76s	0.29s	1.12s	1.07s

Web site	Browser configuration			
	Deprecator		Deprecator + Log	
	Without cache	With cache	Without cache	With cache
55)imdb.com	3.94s	1.99s	11.2s	67.5s
56)office.com	1.85s	0.42s	3.42s	1.94s
57)blogspot.com	1.90s	0.32s	3.05s	2.63s
58)mail.ru	4.02s	1.50s	5.50s	4.40s
59)amazon.co.jp	3.83s	3.03s	11.0s	8.54s
60)github.com	2.97s	0.94s	5.81s	4.45s
61)apple.com	1.90s	0.57s	6.93s	5.38s
62)microsoftonline.com	/	/	/	/
63)pinterest.com	1.49s	0.84s	4.39s	4.92s
64)diply.com	3.01s	1.77s	4.91s	5.34s
65)tianya.cn	3.29s	1.75s	16.4s	2.90s
66)popads.net	1.92s	0.79s	2.35s	1.51s
67)xvideos.com	1.52s	0.69s	8.42s	4.94s
68)wikia.com	2.59s	1.06s	12.0s	7.93s
69)google.com.tr	0.63s	0.19s	1.23s	0.96s
70)csdn.net	10.3s	2.47s	17.0s	12.1s
71)google.com.au	0.64s	0.30s	1.23s	0.99s
72)service.tmall.com	5.04s	1.0s	6.81s	2.21s
73)alipay.com	9.06s	3.02s	12.2s	7.19s
74)google.com.tw	1.04s	0.18s	1.20s	1.04s
75)whatsapp.com	1.35s	0.64s	3.55s	1.74s
76)paypal.com	1.79s	1.05s	4.88s	3.40s
77)xhamster.com	1.59s	0.59s	5.21s	4.44s
78)adobe.com	3.39s	1.97s	10.4s	8.55s
79)youth.cn	10.5s	2.70s	14.0s	7.94s
80)pixnet.net	5.27s	3.43s	19.5s	15.3s
81)soso.com	2.77s	0.86s	4.41s	5.01s
82)coccoc.com	4.85s	2.67s	8.80s	8.02s
83)txxx.com	4.23s	1.40s	16.3s	13.9s
84)google.pl	0.65s	0.18s	1.23s	0.97s
85)dropbox.com	2.57s	1.06s	8.62s	6.99s
86)bongacams.com	3.31s	1.41s	9.91s	8.19s
87)amazon.de	2.97s	1.02s	11.0s	9.13s
88)login.tmall.com	5.56s	1.09s	6.63s	2.60s
89)googleusercontent.com	/	/	/	/
90)porn555.com	4.79s	2.35s	44.6s	33.5s
91)google.co.th	0.65s	0.31s	1.23s	0.98s
92)google.com.eg	0.67s	0.18s	1.32s	0.97s
93)google.com.sa	0.99s	0.18s	1.33s	0.99s

Web site	Browser configuration			
	DEPRECATOR		DEPRECATOR + Log	
	Without cache	With cache	Without cache	With cache
94)fc2.com	1.31s	0.88s	4.79s	3.33s
95)google.com.pk	0.49s	0.20s	1.27s	1.04s
96)china.com	6.30s	3.21s	10.5s	6.87s
97)bbc.co.uk	2.55s	1.14s	6.72s	15.7s
98)craigslist.org	2.03s	1.05s	4.91s	2.67s
99)espn.com	4.50s	1.72s	13.1s	11.0s
100)soundcloud.com	3.08s	1.01s	9.83s	7.09s

Table B.1: Load time of the first 100 site of the Alexa ranking with DEPRECATOR

Web site	Browser configuration			
	Chromium		Chromium without sandbox	
	Without cache	With cache	Without cache	With cache
1)google.com	0.46	0.18	0.26	0.18
2)youtube.com	3.67	2.69	3.32	2.31
3)facebook.com	1.92	1.29	1.89	1.02
4)baidu.com	3.98	0.54	5.01	0.53
5)wikipedia.org	0.88	0.18	0.56	0.17
6)yahoo.com	3.31	2.09	3.28	2.55
7)reddit.com	2.65	1.94	2.05	2.10
8)google.co.in	0.42	0.38	0.26	0.31
9)qq.com	10.4	9.66	9.67	3.75
10)taobao.com	19.3	1.73	4.05	1.51
11)amazon.com	6.20	4.01	5.09	4.67
12)twitter.com	1.18	1.09	1.26	1.22
13)google.co.jp	0.45	0.34	0.46	0.33
14)tmall.com	6.10	1.91	5.16	1.56
15)sohu.com	20.2	13.5	17.7	13.5
16)live.com	4.65	0.53	4.29	0.67
17)vk.com	1.99	0.74	1.96	0.56
18)instagram.com	1.59	0.74	1.54	0.81
19)sina.com.cn	23.0	13.6	17.6	14.3
20)360.cn	9.52	3.46	9.11	2.19
21)jd.com	9.02	1.36	6.89	1.68
22)google.de	0.69	0.33	0.71	0.35
23)google.co.uk	0.68	0.46	1.19	0.41
24)linkedin.com	1.64	0.49	1.18	0.53

Web site	Browser configuration			
	Chromium		Chromium without sandbox	
	Without cache	With cache	Without cache	With cache
25)weibo.com	8.93	3.30	9.60	2.89
26)google.fr	1.14	0.33	0.65	0.31
27)google.ru	0.67	0.22	0.69	0.40
28)google.com.br	1.12	0.34	1.18	0.54
29)yandex.ru	2.04	1.04	2.11	1.10
30)yahoo.co.jp	8.20	4.45	8.87	3.93
31)netflix.com	1.91	0.67	1.54	0.84
32)google.com.hk	0.69	0.35	1.01	0.36
33)t.co	0.35	0.19	0.35	0.20
34)hao123.com	9.65	4.49	9.40	3.08
35)imgur.com	2.74	1.20	3.19	1.00
36)google.it	0.50	0.37	0.47	0.32
37)ebay.com	2.62	1.67	2.42	1.17
38)pornhub.com	2.51	1.95	2.51	1.62
39)google.es	0.66	0.45	0.68	0.41
40)detail.tmall.com	3.37	1.16	3.77	1.24
41)wordpress.com	22.3	7.35	2.03	0.88
42)msn.com	4.46	1.17	2.63	1.47
43)aliexpress.com	3.64	1.50	3.46	1.37
44)bing.com	0.41	0.70	0.46	0.65
45)tumblr.com	6.06	1.69	6.79	1.73
46)google.ca	0.67	0.36	0.97	0.40
47)microsoft.com	1.26	0.61	1.83	0.55
48)livejasmin.com	4.23	2.31	4.06	1.85
49)stackoverflow.com	1.97	1.10	2.12	0.76
50)twitch.tv	2.09	0.69	3.67	1.03
51)ok.ru	2.48	0.71	2.13	0.92
52)google.com.mx	1.00	0.33	0.69	0.39
53)ntd.tv	8.07	6.95	8.50	4.98
54)onclkds.com	0.86	0.25	0.45	0.25
55)imdb.com	4.21	2.48	4.94	2.45
56)office.com	1.80	0.44	1.08	0.44
57)blogspot.com	1.95	0.35	1.66	0.37
58)mail.ru	3.85	1.54	3.84	1.41
59)amazon.co.jp	5.80	2.06	5.59	2.20
60)github.com	3.36	0.80	4.54	0.81
61)apple.com	1.63	0.48	2.24	0.54
62)microsoftonline.com	/	/	/	/
63)pinterest.com	4.15	2.32	3.26	2.35

Web site	Browser configuration			
	Chromium		Chromium without sandbox	
	Without cache	With cache	Without cache	With cache
64)diply.com	4.63	1.46	4.82	1.50
65)tianya.cn	4.36	1.60	9.42	2.22
66)popads.net	1.83	0.33	1.99	1.07
67)xvideos.com	1.59	0.75	1.63	0.60
68)wikia.com	1.86	1.27	3.13	1.37
69)google.com.tr	0.67	0.35	1.11	0.37
70)csdn.net	10.1	2.56	10.1	2.86
71)google.com.au	0.69	0.33	0.78	0.54
72)service.tmall.com	5.06	1.02	3.69	1.42
73)alipay.com	8.75	3.02	12.9	2.98
74)google.com.tw	1.02	0.33	0.80	0.84
75)whatsapp.com	1.21	0.46	1.22	0.32
76)paypal.com	1.52	1.24	1.60	0.80
77)xhamster.com	1.52	0.51	1.65	0.62
78)adobe.com	3.43	1.37	3.43	1.59
79)youth.cn	11.4	3.39	11.1	2.21
80)pixnet.net	13.9	4.34	10.6	4.39
81)soso.com	2.46	0.86	2.81	0.87
82)coccoc.com	5.21	1.69	5.68	1.74
83)txxx.com	3.92	1.17	3.80	1.11
84)google.pl	0.65	0.36	1.22	0.20
85)dropbox.com	3.90	1.75	2.88	1.97
86)bongacams.com	3.46	1.57	4.16	1.59
87)amazon.de	4.85	1.55	5.26	1.82
88)login.tmall.com	5.20	1.14	4.59	1.04
89)googleusercontent.com	/	/	/	/
90)porn555.com	4.95	3.71	5.17	3.32
91)google.co.th	0.67	0.34	1.21	0.20
92)google.com.eg	0.95	0.33	1.84	0.32
93)google.com.sa	0.98	0.33	0.94	0.18
94)fc2.com	2.75	0.93	3.03	1.06
95)google.com.pk	0.71	0.33	0.93	0.27
96)china.com	6.46	3.14	10.6	2.85
97)bbc.co.uk	2.03	1.20	8.52	1.56
98)craigslist.org	2.05	0.48	2.00	0.49
99)espn.com	9.21	6.60	9.19	4.49
100)soundcloud.com	3.71	2.81	3.88	1.74

Table B.2: Load time of the first 100 site of the Alexa ranking with Chromium