Degree Final Project

Uncovering obfuscated web tracking

Computing Specialty

Alvaro Espuña Buxó

Director

Pere Barlet Ros

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

October 16, 2016

Abstract

Several studies showed the prevalence and pervasiveness of user tracking and fingerprinting on the Internet. Since most tracking methods rely on JavaScript, some recent works proposed HTML/JavaScript code analysis as the most effective way to reliably detect user tracking and fingerprinting. The research hypothesis of this project is that web tracking is becoming obfuscated, and previous detection methods based on static code analysis are becoming ineffective and very easy to evade. The objective of this project is to develop a framework based on dynamic code analysis to track the actual calls made to the browser JavaScript API and compare them to the original HTML/JavaScript code in order to detect obfuscated fingerprinting. We will perform a broad analysis of the actual deployment of obfuscated web tracking on the Internet, and release the resulting framework and datasets to the web research community under an Open Source license.[1]

[1]The source code and the gathered data can be found at https://bitbucket.org/aespuna/canvas-tfg

Contents

1 Introduction

This chapter contextualizes the project reviewing the current state of web tracking and some JavaScript obfuscation techniques. It also explains the motivation of the work and presents the structure used on the whole document.

1.1 Context

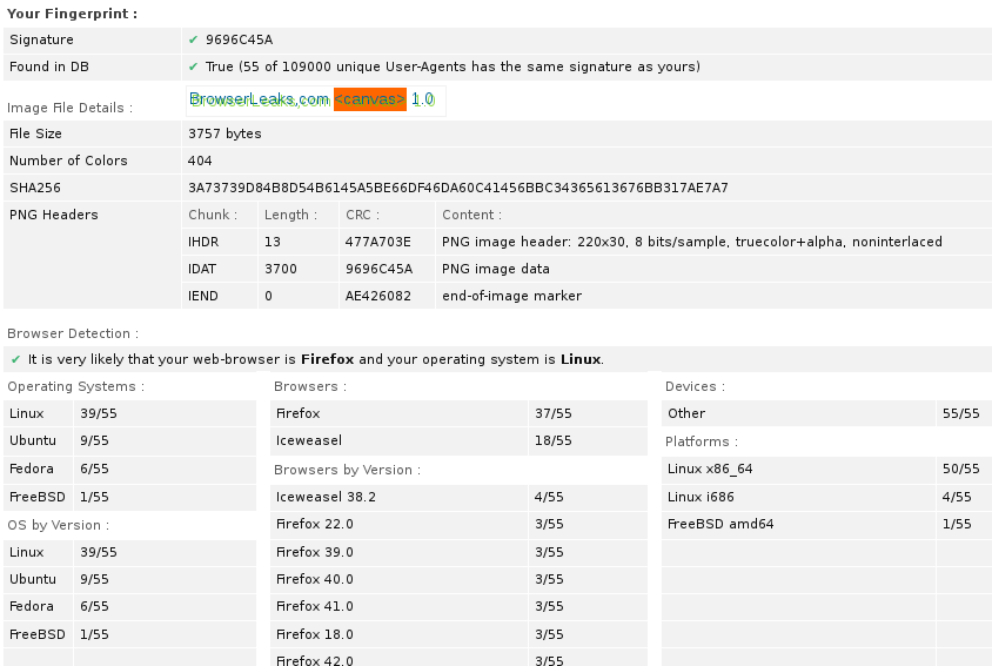
1.1.1 Web tracking

The prevalence of user tracking on the internet is widely known[1]. There are different techniques, but most of them rely on the use of JavaScript because it can be executed in the browser and can be easily deployed. The principal method consists in generating some kind of *digital fingerprint* [2] on the client side to uniquely identify the user, allowing the service to track their movements on the web.

The current JavaScript API[3], implemented by most modern browsers, allows to programatically obtain a lot of information about the client machine. Parameters such as operative system, platform, installed fonts, or

screen dimensions among others can be easily extracted executing simple function calls.

One notable technique is called canvas fingerprinting. The idea is to use a hidden HTML5 canvas element [4], draw some elements using specific fonts, text and shapes, and then produce a raw image using the `toDataURL` method[5]. The corresponding rendered value depends heavily on the client hardware and software and can generate a digital fingerprint (hence the name) that can be used to almost uniquely identify the user.



Your Fingerprint :

Signature	✔ 9696C45A
Found in DB	✔ True (55 of 109000 unique User-Agents has the same signature as yours)
Image File Details :	BrowserLeaks.com <canvas> 1.0
File Size	3757 bytes
Number of Colors	404
SHA256	3A73739D84B8D54B6145A5BE66DF46DA60C41456BBC34365613676BB317AE7A7

PNG Headers	Chunk :	Length :	CRC :	Content :
	IHDR	13	477A703E	PNG image header: 220x30, 8 bits/sample, truecolor+alpha, noninterlaced
	IDAT	3700	9696C45A	PNG image data
	IEND	0	AE426082	end-of-image marker

Browser Detection :

✔ It is very likely that your web-browser is **Firefox** and your operating system is **Linux**.

Operating Systems :	
Linux	39/55
Ubuntu	9/55
Fedora	6/55
FreeBSD	1/55
OS by Version :	
Linux	39/55
Ubuntu	9/55
Fedora	6/55
FreeBSD	1/55

Browsers :	
Firefox	37/55
Iceweasel	18/55
Browsers by Version :	
Iceweasel 38.2	4/55
Firefox 22.0	3/55
Firefox 39.0	3/55
Firefox 40.0	3/55
Firefox 41.0	3/55
Firefox 18.0	3/55
Firefox 42.0	3/55

Devices :	
Other	55/55
Platforms :	
Linux x86_64	50/55
Linux i686	4/55
FreeBSD amd64	1/55

Figure 1.1: This webpage has correctly guessed our user-agent using canvas fingerprinting, even after spoofing our User-Agent HTTP header

Because of how HTTP works, most of the time this data is gathered without the user noticing or without giving explicit consent.

Tracking can also happen when automatically requesting resources from other domains (such as images, scripts or frames) using the HTTP `Referer` Header and already set *cookies*.

To allow inter-domain tracking, that heavily reduces the users' privacy, there are services offered by third party companies that work by deliberately including some HTML element (most of the time a JavaScript snippet) to the first party server response [6].

The gathered information constitutes the core business of several internet advertising companies. They can use it for targeting ads, price discrimination [7] [8] [9], assessing health conditions or even assessing financial credibility[10] [11].

This kind of tracking is not only considered a threat to users' privacy, in some cases it can generate performance issues on the visited website, degrading user experience.

1.1.2 Code obfuscation

Code obfuscation is the process of modifying the source code so it is very difficult to read by a human and consequently reason about it. Since JavaScript must be delivered in its source code form, not in binary form, some content providers obfuscate it to protect it from being understood by others. This way it is more difficult to reuse it, modify it, and sometimes it even helps to prevent copyright violations[12].

Obfuscation must not be confused with minification. Minification tries to reduce the code size (e.g. mangling variable names, avoiding new lines, removing unnecessary spaces), while obfuscation tries to hide the intention of the code. Sometimes obfuscation can lead to code minification or viceversa, but technically they are different concepts.

There are a lot of both free and non-free online tools for obfuscating JavaScript source code[13][14][15][16][17].

```
function divide_by_two(value) {
  var y = value/2;
  return y;
};
divide_by_two(2);

// minified
function f(a){return a/2};f(2)

// obfuscated 1: we can still see the function name
// but it's not near the actual call
eval(function(p,a,c,k,e,d){while(c--){if(k[c]){p=p.replace(new
↪  RegExp('\\b'+c+'\\b','g'),k[c])}}return p}('4 3(0){5
↪  1=0/2;6
↪  1};3(2);',7,7,'value|y||divide_by_two|function|var|return'.split('|')))

// obfuscated 2: divide_by_two doesn't appear
// but the function exists with the same name after
↪  execution
var a={}, b
↪  =["\x64\x69\x76\x69\x64\x65\x5f\x62\x79\x5f\x74\x77\x6f"];a[b[0]]
↪  = function(c){return c/2;};a[b[0]](2);
```

Figure 1.2: Simple example of code minification and obfuscation

The dynamic nature of JavaScript allows to fuse strings with code very easily. Using the `eval` function on a *XOR*ed string with a key, is one of the most used techniques.

One can even hide the `eval` call using the `Function` constructor like [18]

```
[]["sort"]["constructor"](CODE)()
```

Other simpler options include writing the strings simply as hex encoded strings (as the last example in Figure 1.2).

As we can see in Figure 1.2 static code analysis would be useless on obfuscated code.

Our line in differentiating both minification and obfuscation will be JavaScript API name mangling. Minification will not rename public function and variable names, because otherwise the browser would not recognise them. So, if a public function call is being executed without the name being present, it will be considered obfuscation. Otherwise it is just minification.

1.2 State of the art

There are multiple studies and tools around web tracking. Both to perform it, and to protect against it.

In this section we will describe some previous work around web tracking detection and analysis.

There have been some efforts to use software solutions to protect users' privacy on the internet. Most of them take the form of web browser plugins. Notable examples are NoScript [19] (a JavaScript blocker), AdBlock [20] and uBlock [21] (block requests to known domains used for tracking and advertising), and RequestPolicy [22] and Ghostery [23] (block requests to external non-trusted domains). These plugins do not try to detect tracking on the fly, they just basically read user preferences to blacklist some of the content.

1.2.1 Static analysis

There have also been some studies to find malware using static analysis of JavaScript code [24] [25]. It works by analyzing the code and trying to find fragments of it that are used to track web users. Using heuristics and

other classification techniques it determines with only static patterns if it is malware or not.

Detecting malware is a different problem than detecting canvas fingerprinting, but some of this techniques would be applicable.

Some previous research tried to detect code obfuscation automatically statically analyzing the source code [26]. They found that there exists only a weak correlation between malware and JavaScript code obfuscation or minification.

An static analysis approach to search canvas fingerprinting JavaScript function calls would not work if the code is obfuscated, so we will have to execute the JavaScript to really know which are the side effects.

The research hypothesis of this project is that web tracking is becoming obfuscated using tools such as the ones used in Figure 1.2. Because this code transformations happen during execution, previous work using static analysis to detect web tracking are becoming ineffective and very easy to evade.

1.2.2 Dynamic analysis

We have to differentiate between dynamic analysis of obfuscation and dynamic analysis of tracking.

We don't consider dynamic analysis of obfuscation because it is mostly trivial. Once the code has been evaluated, there is no obfuscation. We would only need to compare gobal table symbols after parsing and after evaluating.

On the other hand, dynamic analysis of tracking has not been widely used. After this project started, a mass scale online tracking measurement was done[27].

They didn't take into account obfuscation, but they analyzed a lot of the currently used techniques for tracking, with a framework they opensourced[28].

Those techniques go from different forms of fingerprinting (canvas, WebRTC, fonts, etc) to analyzing third party and cookie usage.

We will try to validate their work using our own implementation of a canvas fingerprinting detection framework, but we will take into account obfuscation deployment.

1.3 Personal motivation

As a internet user I'm worried about the state of tracking on the internet. Not only the reasons behind it but also how most users are completely unaware of it. Learning how the main tracking techniques work, how are they being implemented and how to prevent them it's a reason good enough to take the project.

I am also worried about an obfuscated web. With `WebAssembly`, although it will increase performance and it will allow developers to program in different languages than JavaScript, users won't be able too look at the guts of what's being executed on their own browser. I personally believe in the open web so people can tinker and come up with different solutions and approaches to an already solved problem.

I was interested also in undertanding better how a browser works in a more fundamental basis. The JavaScript engine of a modern browser

is really a feat of engineering, composed of a lot of complex parts. The interpreter, the JIT compiler, the optimizer, ... I expect to learn a lot about them during this project.

1.4 Report structure

This document is separated in multiple chapters, each of them answering a different set of questions:

Scope of the project Presentation of the objectives and explanation of the required knowledge and competences. What are we expecting to do and what not?

Planning How will we distribute the required tasks and budget to obtain our objectives? Will it be sustainable?

Methodology Implementation details of how are we going to achieve our goals. Description of every software element developed during the project.

Results and validation After applying our methodology what are the results? Did we accomplish all the objectives? Can these be considered correct?

Conclusions Why did we obtain what we obtained? What can be changed or added on the future so we can improve our results?

2 Scope of the project

We want to answer if there is obfuscated web tracking on the internet. To achieve this, there are smaller objectives that need to be completed first.

2.1 Objectives

To reduce the complexity of the project we will only focus on canvas fingerprinting as a tracking method. This is the easiest one to instrument, and since it uses the `toDataURL` JavaScript API call, detecting obfuscation will be also doable.

2.1.1 Main objective

The principal objective of this project is to develop a proof-of-concept framework for discovering canvas fingerprinting by analizing the actual calls made to the JavaScript API and comparing them with the original HTML/JavaScript code. This will unveil the obfuscated uses of canvas fingerprinting.

2.1.2 Secondary objectives

Data collection We will create a software tool that will be able to automatically navigate a set of urls and store the results to analize later.

Web service We will create a web service where the user will be able to input an url and our software will visit that webpage and will provide information to the user about the tracking and obfuscation detected on that page. This will help to raise awareness.

Open data We want to be as open as possible. That's why all the data and source code will be released to the public using an Open Source License, so the research community will be able to use it.

2.2 Scope

This project tries to achieve the objectives presented in the section 2.1, but this project is purely research driven. Our work hypothesis is that there is obfuscation of tracking on the internet.

If the answer happens to be false, we will still answer the question and we will try to explain the reasons, so it wouldn't be cosidered a failure.

In any case the tooling required to determine if a certain website uses tracking, and if the tracking is obfuscated will be developed, and will be possible to upgrade it in the future.

2.3 Useful courses

Some useful courses that I took on my specialization are:

Machine Learning (APA)

The knowledge needed in statistics and probability helped to create a correct model to classify the websites

Programming Languages (LP) and Compilers (CL)

Knowledge about lexers, parsers and grammars will be very useful when dealing with the JavaScript and HTML parsing, and finally

Algorithms (A) and Cryptography (C)

Knowledge about hash functions and their implementations helped in finding alternative solutions to consider unique pages.

Some parts of the project, such as the web service or even programming the tooling can be considered transversal competences, and probably would fit better in another specialty. But given that almost all of my optative credits come from the IT specialization it didn't come as a problem.

2.4 Competences

As explained in previous sections, this project uses competences associated to the Computing specialty.

CCO1.2 *To demonstrate knowledge about the theoretical fundamentals of programming languages and the associated lexical, syntactical and semantic processing techniques and be able to apply them to create, design and process languages.*

We won't write a javascript engine. But we need to instrument one to record every call and track the stacktrace, and understanding how it works is important, so this competence can be considered.

CCO1.3 *To define, evaluate and select platforms to develop and produce hardware and software for developing computer applications and services of different complexities.*

We have to deal with the browser, which is a complex piece of software. We will develop a framework that will use a lot of different technologies. It will run locally but also in a dedicated server. These systems are very heterogeneus on platforms and have different requirements.

CCO2.3 *To develop and evaluate interactive systems and systems that show complex information, and its application to solve person-computer interaction problems.*

The cloud service will offer a nice interface for our users. The information we will be giving is critical for them, since they want to know if a certain website is using tracking techniques and moreover if they are being obfuscated. The way we will present this information is important.

CCO2.5 *To implement information retrieval software.*

The core of our system will scrap an important part of the internet just to answer our initial hypothesis. Our whole framework will be an automatic information retrieval software.

2.5 Stakeholders

There are several stakeholders involved in the development of this project, either directly or indirectly.

Developer Is the same person writing this document. His function is to develop the choosen methodology, to find or develop the required tools to achieve the objectives and to validate the obtained results.

Project director The project director is Pere Barlet Ros. His function is to help the developer giving directions on critical points.

Open source community They provide documentation and tools that help on the implementation of the framework. They receive the resulting framework as it will be licensed under the MIT License.

Target audience Is both the research community and the average internet user. One objective of this project is to raise awareness about the ubiquity of internet tracking, and if obfuscation is found, to alert them.

3 Planning

The purpose of this section is to describe the division of the project in smaller tasks. Each task consists of a name, a description, an expected duration, a set of risks and possible complications, task dependencies, and nedeed resources.

All this information will be used to construct a viable action plan that will allow us to finish the project in time, reducing the possibility of failure.

The time planning has been conceived under the assumption that the project was started on September 10th, 2015 and its original deadline was January 25th, 2016.

Due to an unexpected oportunity to grow professionally, I had to decide to park the project for a time. It was this summer that I had enough time to work the hours that this project required.

At the end, it only added a time shift of about 7 months. So *Software developement* and *Results and validation* phases took the expected time, but were done 7 months later.

This was a big problem at the begining, but one advantage is that allowed us to evaluate the work of others[27].

3.1 Tasks

3.1.1 Project preparation

This task consists of choosing the topic and the director to make the project suitable for the Computation specialty. It also includes doing all the paperwork required by FIB, to enroll the TFG.

It took some email exchanging with the director during summer and was already finished at September 14th, 2015.

3.1.2 Project management (GEP) course

This tasks includes a series of smaller tasks that aim to lead the project in the correct direction. It consists of various deliverables and reading some provided documentation.

The expected duration was of 120 hours. The risk level was very low given that the direction was very specific and the schedule was rigid enough. The only task dependency is the project preparation.

The resources used are primarily software: *Atenea* virtual campus, *LibreOffice Writer* to complete the auto-evaluation rubrics, and LaTeX and *emacs* text editor to compose the deliverables.

3.1.3 Topic research

This task is the first project-specific one. It can be divided in two different parts:

1. Read code documentation and bibliography to create a JavaScript tracing framework.
2. Research on various techniques to improve the framework.

This division allowed us to work in a cyclic manner between the software developement of the first part and the topic research of the second.

The expected duration was 90 hours. The first part was completed in parallel with GEP. The only risk was not understanding the papers and spending more time researching for new bibliography.

The resources used were time and internet access.

3.1.4 Approach decision

After researching about different methods to instrument a web browser or different techniques to classify obfuscated and unobfuscated code, we need to decide which ones are the best options for us to implement both in correctness and considering time constraints. This tasks includes meetings with the director.

The expected duration is 25 hours. The risk is being unable to decide a clear winner, or choosing the wrong option so it must be corrected later.

No special resources are needed for this task.

3.1.5 Software developement

This is one of the most important and riskiest tasks. This task will be an iteration over the framework we aim to develop, improving it iteration over iteration. The resources used will be two computers (one to develop software, the other to run the public web server) and a text editor to code. This task has a dependency on the approach.

The first step would be to make the JavaScript tracing framework, probably with the aid of a modern web browser Javascript engine. The expected duration is 40 hours. The risk is being unable to work with the complexity of the code of a modern browser (state of the art JIT compiler, micro optimizations, etc).

The second step would be to write a program that uses the data collected by the tracing framework so it can determine if there is code obfuscation or it is being used for tracking. The more sofisticated we want this software to be, more time it will take. The expected duration is 60 hours. This part has a dependency on the approach decision.

A third step would be to create a web service to use the obfuscation and tracking system remotely. This should be the less riskier part, and the expected duration is 20 hours.

3.1.6 Results and validation

This task consists of analyzing the data obtained by the framework to check its correctness manually. This will be used to improve it in future iterations. This composes a cycle with software development task, but can be represented linearly. The expected duration is 80 hours.

3.1.7 Public release

This task consists of releasing the data and results collected and the source code developed during the project (browser modifications, tools and server) to the public domain. Some formatting and cleaning up may be needed, to make it easier to read and use for the community.

The expected duration is of 10 hours. This task has as a dependency the results and validation task and has low risk.

The resource used will be some code sharing tool such as `github.com` or `bitbucket.org`.

3.1.8 Memory writing

A memory must be written explaining the whole project. Although after the GEP course, almost a 70% of the memory will be already drafted, there are some time consuming things remaining such as the final polishment of already written parts, writing of the technical sections or proof-reading the whole document multiple times.

The expected duration is of 30 hours. This a very important part of the project. This task has as prerequisite the results and validation task.

The used resources will be *emacs* and *LaTeX*.

3.1.9 Oral defense preparation

After the project memory has been delivered, ther is one week before the oral defense. This task includes preparing sides and practicing the defence. The expected duration is of 15 hours.

3.2 Gantt diagram

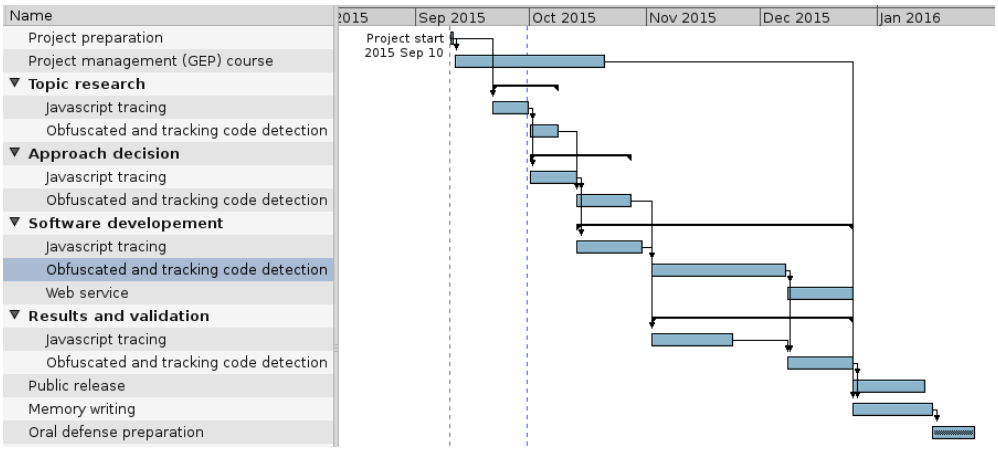


Figure 3.1: Original Gantt diagram representing all tasks and its dependencies

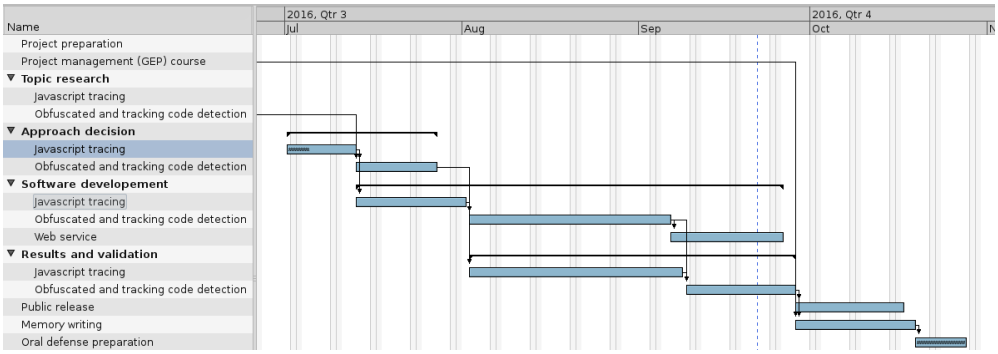


Figure 3.2: Final adjusted Gantt diagram

3.3 Risks and alternatives

During the project, short after the GEP task finished we experienced an unexpected major delay. I was offered an opportunity that I wasn't able to refuse. This experience helped me to grow both personally and professionally, but at the cost of having to park the project until this summer.

At the end it only concurred a general time shift, and both taskwise duration and total budget were unaffected. The total time of the project obviously expanded.

However, taskwise, the previous planification still worked. And we had a plan for every task if a delay was detected, we considered diferent levels of correction according to the severity:

less than 3 days There is no need to apply correctional measures, as there is a margin of 2-3 days.

3 to 10 days We increase the daily dedication hours by 10% - 25% until the deviation for that task is compensated.

more than 10 days We have to rethink and weight how important this task is. If there are a lot of dependencies we will have to finish this task no matter what, and then select the most important subsequent tasks. This could dramatically decrease the quality of the final product.

3.4 Budget estimation

We classify resources according to their nature: hardware, software, human, and utilities. We will see that hardware and software resources have negligible cost for various reasons, and that the largest part of the budget falls in the human category.

3.4.1 Hardware resources

The hardware resources used for this project consist of a laptop for the developement of the needed code, and a computer that will be used as a server to run the cloud solution to detect obfuscation.

I will be using my laptop, which I bought for 275€ a year ago. This cost can be excluded from the project as any other computer could serve the same purpose.

The web service will be hosted in a Virtual Private Server provided by ovh.es. The economic cost is around 6€ per month. We don't expect to use a lot of cpu, and since it's a virtual server, the electrical consumption will not effectively increase.

The electricity cost might amount to a total of 15€ for the duration of the project, which can be considered negligible.

3.4.2 Software resources

All software used will be free, including but not limited to: Operating System, text editor, browser, and pdf reader. These programs, are not only free in price, but also in its licensing.

If developing a browser plugin is required, there's no need to test it in other Operative Systems, since the browser provides the required abstraction.

For releasing the code to the public, a free cloud platform such as Github[1] or Bitbucket[2] will be used. These platforms allow free hosting for open source projects.

In conclusion there are no software costs.

3.4.3 Human resources

These, as explained earlier, compromise the bigest part of the budget. Table 3.1 shows the cost for every role that this project needs.

Role	Payment (€/h)	Hours	Total (€)
Project manager	50	70	3500
Software designer	35	80	2800
Software programmer	25	145	3625
Software tester	20	120	2400
Total		415	12325

Table 3.1: Human resources costs per role

3.4.4 Total budget

Using the estimation of costs for every resource we obtain a required budget no greater than 12400€.

[1] https://github.com
[2] https://bitbucket.org

3.5 Budget control

As we have seen in the previous section, the budget is mainly for the human resources. Modifications are very improbable because the project is expected to be doable with very limited hardware specifications and with free software. Variations could be introduced by increasing the time needed to finish the project.

Another factor could be an unexpected success of the unobfuscator service, which would increase the cost of the server or, more importantly the maintenance cost of the product.

In conclusion, the budget is low and very stable.

3.6 Sustainability

In this section we are going to evaluate the sustainability of our project in three different areas: economic area, social area and environmental area.

3.6.1 Economic sustainability

Since we are relying on free and open source libraries and utilities that depend on foundations and self maintained institutions, the economic sustainability of the project can be guaranteed.

This project is relevant because as we will see can be provide a huge social impact and it is innovative in the way to process tracking on the web. This justifies the project.

The project receives an 7 in the economical viability area because its costs cannot be further optimized, but given that the success of it relies on that there is enough obfuscation of tracking on the web, there is a certain degree of risk.

3.6.2 Social sustainability

Although an important part of the public knows that there is tracking on the web going on, proving that a given service is obfuscating it might create a huge impact. Also, we expect some impact inside the computer security research community.

This project receives a 8 in the social sustainability area, since its importance can be huge for the general public, and can affect people's point of view.

3.6.3 Environmental sustainability

The project requires minimal resources. We estimate a monthly average energy consumption of 20kWh, which is equivalent to approximately 15kg of CO_2.

This could increase if the cloud service is a huge success, but *a priori* the electricity consumption of the server will not be increased significantly just by running our instrumented browser.

When possible, we will avoid to print documents on paper and use digital versions instead.

This project is awarded a 9 in the environmental sustainability area because it does not use unnecessary resources, and the required ones are used in a responsible and efficient manner. Its footprint for the environtment is very low.

4 Methodology

The methodology we agreed upon with the director consists on a **Firefox plugin** that will connect to a running local server, logging every *JavaScript* call to the `toDataURL` function of the `HTML5 Canvas element`. It will log information such as display dimensions of the *canvas*, the full stacktrace to that call and the current url. We try to detect only canvas fingerprinting since it's the easiest of the tracking methods to instrument.

The **logger server** will process this information and store it in a structured database, so it can be queried and analyzed later.

The **scraper** will run an stock version of Firefox with our plugin installed. It will use the Alexa's[1] list of the top 1 million most visited pages[2], although we will only care about the first 10000. This sample size should be enough to find if there's obfuscation in mainstream websites. The scraper should be implemented in a modular and flexible fashion, so future students and researchers can add more interactivity with the visited webpages such as filling forms or clicking certain buttons. We will be using the *Selenium WebDriver*[3].

[1] http://www.alexa.com/topsites
[2] http://s3.amazonaws.com/alexa-static/top-1m.csv.zip
[3] http://www.seleniumhq.org/projects/webdriver/

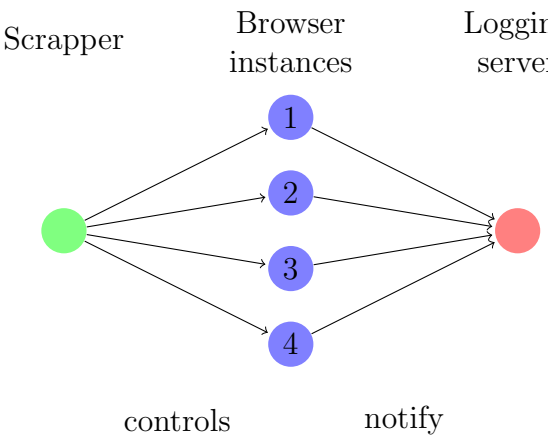


Figure 4.1: Architecture schema

We will also implement a web service so a visitor can introduce an URL in a simple form and using this same methodology we will be able to tell the user if that webpage implements canvas fingerprinting and if it's obfuscated.

If we ran the logger in a public server, and made the plugin send its request to that server, we could distribute the plugin through the Mozilla addon store and crowdsource the collection of the data more easily. We won't do this because of time and budget limitations. We wouldn't be able to achieve enough diffusion to make it relevant enough for this project.

4.1 Browser plugin

The plugin basically injects a modified version of the `toDataURL` method to every webpage the browser visits. It tries too reduce its footprint removing itself from the *DOM* after the injection. This will give us more accurate stacktrace positions.

There have been some modifications on the original idea. We now collect the canvas **visible** size (`scrollHeight` and `scrollWidth`) so we

can filter out false positives. We have to take into account that not all canvas elements are used for tracking, it has its legit purposes. If the canvas is smaller than a square with side 16px, we can consider it is being hidden from the user[27].

Also, we send a *snapshot* of the whole `document` to the server in the moment of the execution. This way we can filter out unobfuscated calls that are injected to the *DOM* by third parties. If we analyzed the `HTML document` before any *JavaScript* execution (using a proxy such as `mitmdump`[4] for example) we could potentially have a very high false positive rate. This presents a major advantage over the original idea.

Our modified `toDataURL` does the same thing as the original function (to keep *compatibility*), but also raises and captures an exception, so we can get a stacktrace. Then, the information about the canvas and the call is `POST`ed to our logging server (normally listening on `localhost`).

The main gist of the plugin is:

```
var scriptNode = document.createElement('script');
function instrument() {
  var old = HTMLCanvasElement.prototype.toDataURL;
  HTMLCanvasElement.prototype.toDataURL = function(c) {
    var trace = (new Error).stack;
    var xhr = new XMLHttpRequest();
    xhr.open("POST", our_server_address, true);
    xhr.setRequestHeader("Content-Type", "application/json");
    var params = {
      w: this.scrollWidth,
```

[4] http://docs.mitmproxy.org/en/latest/mitmdump.html

```
      h: this.scrollHeight,
      referrer: document.referrer,
      src: window.location.href,
      stack: trace,
      doc: new XMLSerializer().serializeToString(document),
    }
    xhr.send(JSON.stringify(params));
    return old.apply(this, arguments);
  }
  var self = document.currentScript;
  self.parentNode.removeChild(self);
}
scriptNode.innerHTML = '('+instrument.toString()+')();';
where = document.head || document.body;
if (where) {
    where.insertBefore(scriptNode, where.firstChild);
}
```

We have to note, that if there is no document (`where` is `null` and thus evaluates to false), we are visiting a script that was loaded by an HTML snippet, and our modified version of `toDataURL` will already be set up.

4.2 Logger server

The logger is basically a server that binds to an arbitrary address and port (normally it will be localhost, but this decoupling gives us more flexibility and reusability) and writes it to an SQLite database. We are using this database engine so the collected data can be shared simply distributing one

binary file. Also, since the server is implemented in *Python*[5], that includes a module for SQLite in the standard library, the setup time was very low compared to Postgres or MySQL. Basically it is good enough. The server accesses to the database using an *ORM* so migrating it to another RDBM should be doable in the future without many modifications in the server code if it is required.

Also, since most of the times we are dealing with third parties, we want to detect if different pages using the same third party, are really using the same files/code. Ideally we would hash the contents and then we would be able to compare hashes instead of storing the whole file multiple times.

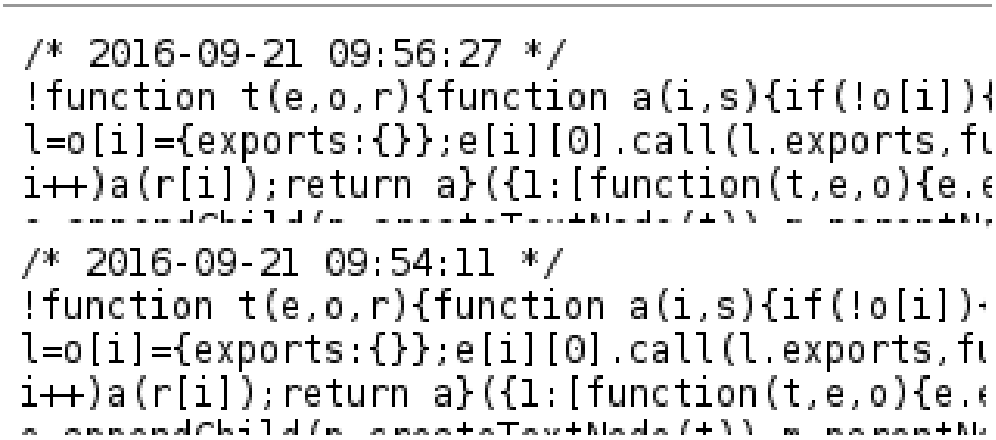


Figure 4.2: These two files only differ in the timestamp

This is not really feasible. Sometimes, there are slight modifications such as timestamps or user agent information, that are included dynamically in the response. These modifications don't change the *overall identity* of the file, but a typical hashing function, like any of the *SHA* family for example, would not work here, because these small changes produce completely different hashes (in fact, that's their purpose). This is why we are using a locality sensitive hashing algorithm (like the one implemented in

[5] https://python.org

tlsh[6][29] to better detect file *equality*.

We decided to use *Python* for the `HTTP` server because it is very flexible, it has a lot of useful libraries and it is very simple yet powerful.

It is important that we implement a valid response with the `OPTIONS` method, because since we are connecting to another domain (from the visited page to ourselves) we want to be CORS[7] compliant[30].

As it can be seen in Figure 4.3 the database schema is very simple. We are just interested in aggregating and storing the data in a structured way. We want to remark that we are differentiating between the caller (where the stacktrace ends) and the initiator (where the stacktrace starts). Storing these makes it easier to detect third parties and who is really responsible of the call. This information could be extracted later from the also stored full stacktrace.

4.3 Scraper

The scraper is also a *Python* script that orchestrates multiple Firefox instances. It uses the Selenium WebDriver module for python, and runs Firefox with our plugin installed.

It then loads a file containing a list of urls and navigates them one after the other.

Because it is already using the Selenium WebDriver, in a future it would be possible to add more functionality such as clicking buttons or filling forms.

[6] https://github.com/trendmicro/tlsh
[7] https://developer.mozilla.org/en/docs/Web/HTTP/Access_control_CORS

```
CREATE TABLE domain (
        id INTEGER NOT NULL,
        domain VARCHAR NOT NULL,
        alexa_rank INTEGER NOT NULL,
        PRIMARY KEY (id),
        UNIQUE (domain)
);
CREATE INDEX ix_domain_alexa_rank ON domain (alexa_rank);
CREATE TABLE log (
        id INTEGER NOT NULL,
        domain_id INTEGER NOT NULL,
        measured_at DATETIME,
        canvas_width INTEGER,
        canvas_height INTEGER,
        referrer VARCHAR,
        source_url VARCHAR,
        source_html VARCHAR,
        source_tlsh VARCHAR,
        stacktrace VARCHAR,
        st_caller_file VARCHAR,
        st_caller_line INTEGER,
        st_caller_char INTEGER,
        st_caller_tlsh VARCHAR,
        st_init_file VARCHAR,
        st_init_line INTEGER,
        st_init_char INTEGER,
        st_init_tlsh VARCHAR,
        is_obfuscated BOOLEAN,
        PRIMARY KEY (id),
        FOREIGN KEY(domain_id) REFERENCES domain (id),
        CHECK (is_obfuscated IN (0, 1))
);
```

Figure 4.3: Database schema

```
def build_driver():
    fp = webdriver.FirefoxProfile()
    fp.add_extension(config.PLUGIN_FILE_PATH)
    # To allow connections from https to our localhost
    fp.set_preference("security.mixed_content.block_active_content", False)
    caps = DesiredCapabilities.FIREFOX
    return webdriver.Firefox(firefox_profile=fp, capabilities=caps)
```

Figure 4.4: This function returns a webdriver that will run Firefox with our plugin already installed

A page visit is considered complete 15 seconds after the browser fully loaded the content. This way we will also catch delayed calls or time triggered events.

The scraper is prepared to be highly parallelizable, but it is not recommended to run a lot of Firefox in a computer with reduced resources.

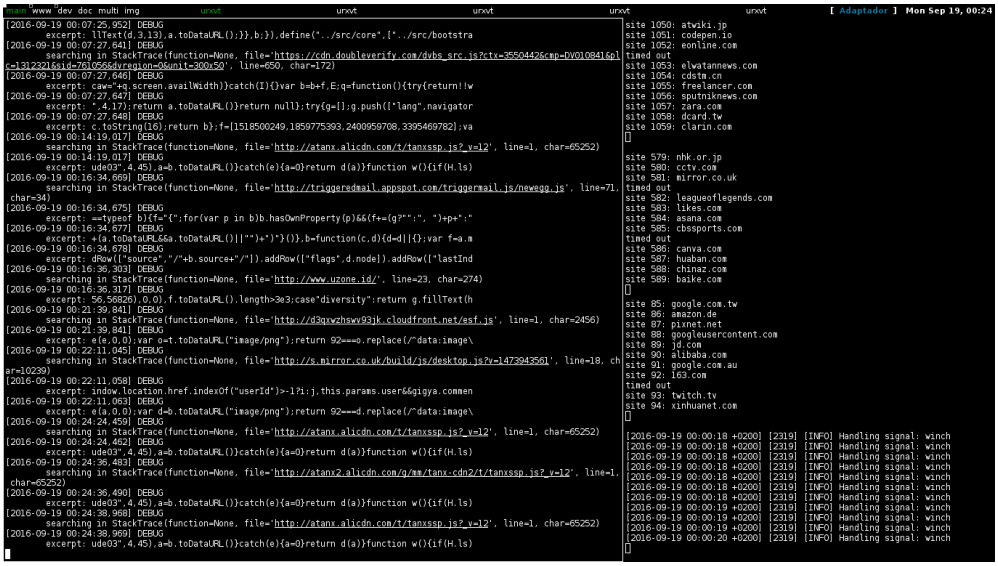


Figure 4.5: Scraper working on my personal computer

4.4 Web service

The web service basically runs on a Virtual Private Server everything we have already described. The only difference is that the scraping process receives urls remotely on demand. Then the result is displayed to the user in an informative and readable format.

Figure 4.6: Webservice result display

To make it work on a virtual private server, without a graphical card and without graphical user interface, some adjustments had to be done.

- The scraper is now a service, listening to what urls to visit.
- The scraper notifies the web service when a certain visit has already finished.
- The scraper will run the Selenium WebDriver as a server to reduce browser launching time.
- The web service caches results up to a day, so frequently requested webpages can have their answer quickly.

- Firefox runs now on the `X` virtual framebuffer (`xfvb`).

Here we can see how implementing the logging server as another service, helped with the modularity. We didn't have to modify it at all, only the behaviour of the client (the scraper).

Although the virtual framebuffer should use more memory than a headless version of Firefox, we chose this solution because we don't expect to have too many traffic, and it is much simpler to implement.

4.4.1 Security considerations

This service is just a proof of concept of how the developed tools could be used in different ways. Allowing our users to make our server visit arbitrary webpages can be a security problem.

Some measures were taken into account, such as:

- The browser is being run by an unpriviliged user.
- This user has very limited resources assigned such as memory, disk space, and cpu.
- We will clear cookies after the visit has finished
- We will kill the browser instance after certain time automatically
- Only 4 instances of firefox will be running at maximum at any given time.
- We will cache the results to avoid accessing the same webpage too many times (although very improbable, we don't want to be responsible of a DoS attack)

With this webservice, pages that require a login, will still be inaccessible for us, since we are only receiving an url as input (not the credentials). This would be solved if we distributed the plugin and made the logging server public, but the problem with this approach then would be that we would have to redact all personal information from the logged `HTML`, making it difficult to know what to redact and what not.

5 Results and validation

After browsing with our framework the landing page of the 10000 most visited webpages according to Alexa, these are the obtained results:

A total of 644 (6.4%) diferent pages use canvas fingerprinting as detected by our methodology. There were 1424 calls detected. Some pages use more than one tracking service, or a single service sometimes needs more than one call to create a full fingerprint.

Comparing the tlsh hashes with a threshold of 75 (out of 400, meaning they are very similar) we get that there are only 90 different files. This is important to stregthen the fact that most of the tracking is controlled by a small fraction of third parties.

The most used tracking services that perform a canvas fingerprint are bktrx.com, doubleverify.com, yandex.ru, alicdn.com, and adnium.com.

Our results are only slightly different compared to the ones obtained in previous studies[27].

We can also see the trend of how less popular pages use less tracking. In the 0-10K range the percentage is about a 5.5%

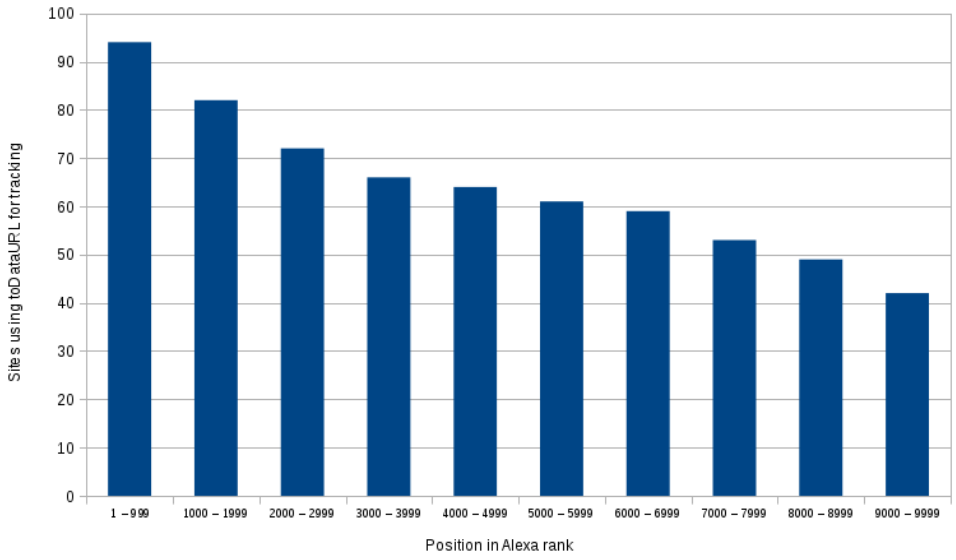


Figure 5.1: Tracking bucketed by Alexa rank position

We also observed that the prevalence of `lijit.com` has gone down in favor of `bktrx.com`, used by Microsoft and some of his affiliates.

There was a total of 542 calls that have been excluded since the canvas have non-zero width and height (in fact it is 16x16 px) and after further manual inspection, it is clear it's only used to test if the device supports emojis provided by the wordpress plugin `https://wordpress.org/plugins/wp-emoji-one/`.

There are only 97 detected obfuscated calls that originated from 33 different pages. Almost all of them are from third parties (the tracking code comes from only 21 domains, the most used being `js.ad-score.com`, `cdn.inaudium.com`, and `fraudmetrix.cn`). The distribution of the rank of the pages that originated the call can be seen in Figure 5.2

It's interesting to see that most of the calls come from external scripts, loaded by the webpage. This being totally dynamic allows much more

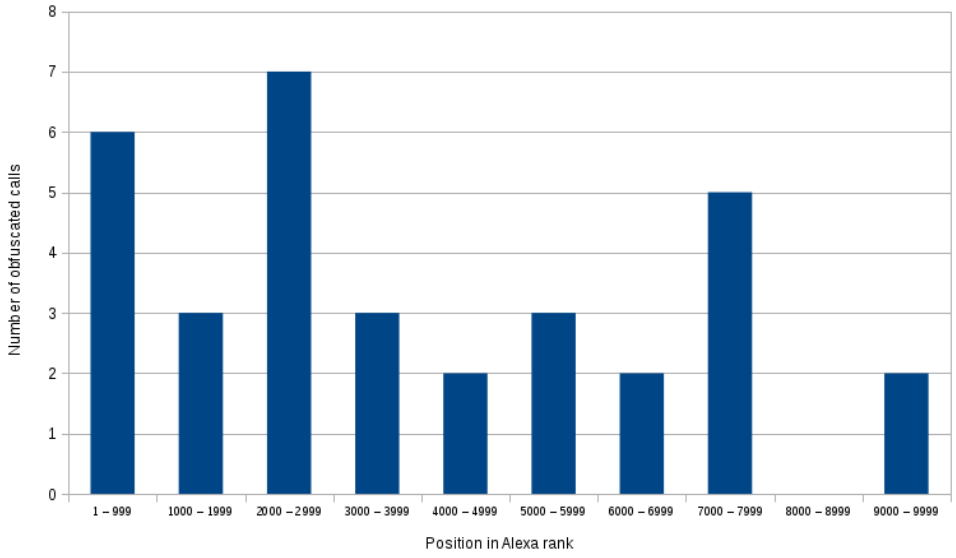


Figure 5.2: Obfuscation bucketed by Alexa rank position

injection control to the advertising provider, making it more dangerous for the final user.

It is also interesting to note that, the most used tracking providers, Google Analytics and doubleclick.net[1] don't use canvas fingerprinting, as it is considered a more intrusive method, and as explained in *Online Tracking: A 1-million-site Measurement and Analysis*[27] recent awareness produced that major advertisers moved away from it.

The whole process of data gathering took almost 20 hours spanning the 17th and 18th of September.

We can conclude that there is obfuscation on only very specific cases. This can be understood taking into account that most users don't really analyze the code, and there exist dynamic tools to avoid it, so the effort spent on obfuscating is probably not *rewarding* enough.

[1]Also owned by Google

```
__pm_glbl = {cfg: {server_token:
↪  "6a186ba29d36faad-GkvfPcVjdVPqKDwT3HHJEE0=-E07GOstkbVbnMg==",
↪  host: "js.ad-score.com"}};(function(){function
↪  ea(){for(var
↪  b=a[0],c=0;24>c;c++)b+=a[2][a[1]](Math[a[3]](62*Math[a[4]]()));return
↪  b}function ba(){return Date[a[26]]?Date[a[26]]():(new
↪  Date)[a[27]]()}
...
a=["",b("kpizIb",18),
b("OPQRSTUVWXYZABCDEFGHIJKLMNopqrstuvwxyzabcdefghijklmn0123456789",12),
b("qwzzc",15),b("mviyjh",5),
b("__ax_rwmw_",15),b("_",24),b("j",17),b("__tq_kpfp",22),b("jmn",19),
b("iuhluh_jeaud",10),b("EQDHQD_FAWQZ",14),b("mtxy",21),
b("ox.fi-xhtwj.htr",21),b("jlini",6),b("eqqmp",3),b("AFPXYIBA_JLARIBP",3),
b("__he_kwjnwj_lgcwf",8),b("eqdhqd_bmdmye",14),b("buklmpulk",19),
b("tds_iti",25),b("chxyrIz",6),b("gifkfkpgv",9),b("tmvobp",18),
...
```

Figure 5.3: Excerpt of https://js.ad-score.com/score.min.js, that is obfuscated

Most webpages delegate the tracking to third parties, and there will be obfuscation only when they care enough and think that it's effective.

The whole database can be downloaded at https://bitbucket.org/aespuna/canvas-tfg/downloads/stats.db.

The number of results and more importantly, the rate is very manageable. At first we validated some of the results by hand. This allowed us to iterate the plugin development, fixing some corner cases, and reducing the number of false positives, without having to repeat the whole process from the begining.

The obtained results are somewhat expected and they overlap with the interpretation of *Online Tracking: A 1-million-site Measurement and Analysis*[27].

6 Conclusion

6.1 Summary

With this project we proved that tracking is very prevalent today on the internet. We looked only for a particular technique, and the relative usage was high enough. Also, even though we discovered some obfuscation, it cannot be considered that it is being used extensively.

Our opinion is that probably it takes too much work and effort to obfuscate something that nobody will look at. Most of the time minification is enough to make it unreadable by a human (but preserving the public calls). The users that don't want to be tracked are probably already using a plugin or a set of browser configurations that allow to detect and avoid being tracked dynamically. That's why probably distributors don't care enough to obfuscate their code. We need to raise awareness, so internet users can know these tools and protect against tracking using them.

So the best option is to use plugins that find these threats in real time, and simply avoid executing certain paths on the client browser. A very good option to protect our privacy is the Tor browser[1], that bundles a lot of privacy protecting plugins.

[1] https://www.torproject.org/projects/torbrowser.html.en

6.2 Future work

We created an extensible framework that can be used by future students. A non-extensive list of possible extensions could be:

- Detect obfuscation even if there's no tracking.
- Add more browsing capabilities such as form filling and button clicking, because maybe we lost a big part of the tracking visiting only the landing page.
- Try detect other tracking techniques (such as *evercookies*, `WebRTC` fingerprinting...) and potentially find obfuscation.
- Distribute the plugin so it can be used by everybody and collect the data in a public server.
- Write more documentation about some of the parts to make it more easy to use.

There are still possible performance updgrades for the web service. But they should be irrelevant once the plugin is publicly distributed.

We could also use our obtained data to make pressure using entities such as the *EFF*[2] to show that canvas fingerprinting is still being used, and help educate the average internet user.

[2]Electronic Frontier Foundation https://eff.org

6.3 Personal thoughts

This project was instructive in multiple ways.

It started as a continuation of a master's final project from another student, but we quickly changed the approach since a big part of the code was already outdated. This new approach simplified every piece of the toolchain, and allowed us get results faster. I had to develop little pieces that were glued together at the end, and that is always satisfying.

Getting to know more about the internals of the web and its ecosystem was something I had interest on, and this project allowed me to work on it.

The only thing I regret is not finishing it earlier due to my professional needs. The 7-months delay was the worst part of the project. I had to adapt and work hard during this summer to be able to finish it. It was mentally exhausting to not be able to spend all the required time at the begining and knowing it would be delayed, but I didn't want to lose the opportunity and the advisor was very understanding about the whole situation.

At the end I can say I'm proud of the result. I hope it will be useful for a future student or researcher.

Bibliography

[1] Tomasz Bujlow, Valentín Carela-Español, Josep Solé-Pareta, and Pere Barlet-Ros. Web tracking: Mechanisms, implications, and defenses. http://arxiv.org/pdf/1507.07872.pdf, 2015. [Online; accessed October 10, 2016; Submitted to IEEE Communications Surveys and Tutorials].

[2] Electronic Fredoom Frontier. Panopticlick: How unique and trackable is your browser? https://panopticlick.eff.org/, 2010.

[3] Ecma International. Ecmascript© 2016 language specification. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf, 2016.

[4] 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 21st acm conference on computer and communications security (ccs 2014), 2014.

[5] Canvas fingerprinting. https://www.browserleaks.com/canvas.

[6] Google. Set up the web tracking code. https://support.google.com/analytics/answer/1008080?hl=en, 2015.

[7] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. Detecting price and search discrimination on the internet, 2012.

[8] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. Crowd-assisted search for price discrimination in e-commerce: First results, 2013.

[9] Aniko Hannak, Gary Soeller, David Lazer, Alan Mislove, and Christo Wilson. Measuring price discrimination and steering on e-commerce web sites, 2014.

[10] Facebook friends could change your credit score. http://money.cnn.com/2013/08/26/technology/social/facebook-credit-score/index.html?hpt=hp_t2, 2013.

[11] Insurance data: Very personal finance. http://www.economist.com/node/21556263, 2012.

[12] Pedro Fortuna. Protecting javascript source code using obfuscation. http://www.slideshare.net/auditmark/owasp-eu-tour-2013-lisbon-pedro-fortuna-protecting-java-script-source-code-using-obfuscation, 2013.

[13] Online javascript obfuscator. http://www.javascriptobfuscator.com/JavaScript-Obfuscator.aspx, 2015.

[14] Protect javascript code – semanticdesigns.com. http://www.semanticdesigns.com/Products/Obfuscators/ECMAScriptObfuscator.html, 2015.

[15] Javascript obfuscate and encoder. http://www.jsobfuscate.com, 2015.

[16] Jscrambler: Protect your javascript. https://jscrambler.com/en/, 2015.

[17] Javascript obfuscator – free online javascript packer. http://packer.50x.eu, 2015.

[18] Martin Kleppe. Jsfuck. http://www.jsfuck.com/.

[19] Noscript - javascript/java/flash blocker for a safer firefox experience! https://noscript.net/, 2015.

[20] Adblock plus - surf the web without annoying ads! https://adblockplus.org/, 2015.

[21] ublock - a fast efficient web ad blocker. https://ublock.org/, 2015.

[22] Firefox addon for privacy and security - requestpolicy. https://requestpolicy.com/, 2015.

[23] Ghostery. take control of your digital experience. https://www.ghostery.com/en/, 2015.

[24] Eunjin (EJ) Jung Peter Likarish. Obfuscated malicious javascript detection using classification techniques, conference: Malicious and unwanted software. http://www.researchgate.net/publication/224110475_Obfuscated_malicious_JavaScript_detection_using_classification_techniques, 2009.

[25] Fangfang Zhang Wei Xu and Sencun ZhuJ. Still: Mostly static detection of obfuscated malicious javascript code. http://www.cse.psu.edu/~sxz16/papers/JStill.pdf, 2010.

[26] Scott Kaplan, Benjamin Livshits, Ben Zorn, Christian Siefert, and Charlie Cursinger. "nofus: Automatically detecting" + string.fromcharcode(32) + "obfuscated".tolowercase() + "javascript

code". http://research.microsoft.com/apps/pubs/default.aspx?id=148514, May 2011.

[27] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. http://randomwalker.info/publications/OpenWPM_1_million_site_tracking_measurement.pdf, May 2016.

[28] Steven Englehardt and Arvind Narayanan. A web privacy measurement framework. https://github.com/citp/OpenWPM/, May 2016.

[29] Jonathan Oliver, Chun Cheng, and Yanggui Chen. Tlsh – a locality sensitive hash. http://ieeexplore.ieee.org/document/6754635/, 2013.

[30] WWW Consortium. Cross-origin resource sharing. https://www.w3.org/TR/cors/, January 2014.