**University of Warsaw**
Faculty of Mathematics, Informatics and Mechanics

**Tomasz Kępa**
Student no. 359746

# Detecting Anti-Adblockers using Differential Execution Analysis

**Master's thesis**
**in COMPUTER SCIENCE**

Supervisor:
**dr Konrad Durnoga**
Institute of Informatics

August 2019

## Supervisor's statement

Hereby I confirm that the presented thesis was prepared under my supervision and that it fulfils the requirements for the degree of Master of Computer Science.

Date Supervisor's signature

## Author's statement

Hereby I declare that the presented thesis was prepared by me and none of its contents was obtained by means that are against the law.

The thesis has never before been a subject of any procedure of obtaining an academic degree.

Moreover, I declare that the present version of the thesis is identical to the attached electronic version.

Date Author's signature

## Abstract

Ads are the main source of income of numerous websites. However, some of them are fairly annoying which causes many users to use adblocking browser extensions. Some services, in turn, use specialized scripts to detect such plug-ins and silently report them or block some content as a punishment. The goal of this thesis is to build a pipeline for detecting such scripts based on a differential execution analysis, a method provided by other authors in 2018. Such a mechanism can be used later to analyze the prevalence of anti-adblockers on Polish websites or to build an extension capable of circumventing such scripts.

## Keywords

dynamic analysis, differential execution analysis, javascript, anti-adblockers, ads

## Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatics, Computer Science

## Subject classification

Software and its engineering. Dynamic analysis

## Tytuł pracy w języku polskim

Wykrywanie skryptów blokujących rozszerzenia typu AdBlock w przeglądarkach

# Contents

# Introduction

TODO:

# Chapter 1

# Basic concepts

## 1.1. Definitions

- Execution event – each occurence of control executing some statement, entering or leaving a control structure etc.
- Execution trace – a series of execution events collected during program execution. It is dependent both on program structure and it's input (also implicit such as randomly generated numbers)
- Execution index
- Stable marriage problem

## 1.2. JavaScript execution model

# Chapter 2

# Related work

## 2.1. Measuring and Disrupting Anti-Adblockers Using Differential Execution Analysis

The method is based on paper by Zhu et al. [2]

# Chapter 3

# Trace collection

## 3.1. Methods overview

There are a few distinct ways to obtain execution trace of JavaScript code.

The simplest (and most limited) methods use only mechanisms present in the language. More elaborate inject special tracing code to analyzed script. The last kind modify the engine to produce the desired data.

## 3.2. Dynamic code instrumentation

JavaScript is a very dynamic language. For this reason it is relatively easy to write code that will modify each function present in the environment to log each entry and exit, possibly along with all the arguments and return value [1].

Snippet **TODO**: add snippet showing how to do that is an example of code that does just that. It replaces function definition with new one that logs its arguments before calling the old function and logs return the return value after that.

It is also possible to traverse recursively the entire global object and replace each function this way.

However, this is not enough for the needs of Differential Execution Analysis. The most obvious limitation is that it's not possible to instrument control statements. Another major shortcoming is that function can be instrumented only after they are defined. It is quite an obstacle, because JavaScript allows to define function practically anywhere and it is very common to use anonymous functions as callbacks. It is not possible to instrument such callback without modyfing the instumented code.

## 3.3. Code rewriting

## 3.4. Engine instrumentation

# Chapter 4

# Trace collection by V8 instrumentation

**4.1. V8 architecture**

**4.2. V8 usage in chromium**

**4.3. Chrome's extensions architecture**

**4.4. V8's *--trace* flag**

**4.5. Bytecode injection**

**4.6. Controlling Chrome programatically**

# Chapter 5

# Trace analysis

## 5.1. Trace untangling using execution index

### 5.1.1. Optimizations

## 5.2. Trace matching using SMP

## 5.3. Noise filtering

# Chapter 6

# Evaluation

# Bibliography

[1] StackOverflow users. Adding console.log to every function automatically. `https://stackoverflow.com/questions/5033836/adding-console-log-to-every-function-automatically`, 2017. [Online; accessed 25-July-2019].

[2] Shitong Zhu, Xunchao Hu, Zhiyun Qian, Zubair Shafiq, and Heng Yin. Measuring and disrupting anti-adblockers using differential execution analysis. In *NDSS*. The Internet Society, 2018.