Université de Lille

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

---

Habilitation à Diriger Des Recherches
Université de Lille

École Doctorale Sciences Pour l'Ingénieur
Spécialité : Informatique

Walter Rudametkin

Université de Lille & CRIStAL
Équipe-Projet Spirals

Improving the Security and Privacy of the Web through Browser Fingerprinting

Soutenue le 21 juin 2021 devant le jury compose par

Rapporteurs:	Sonia Ben Mokhtar	-	Directrice de Recherche CNRS
	Benjamin Nguyen	-	Professeur à l'INSA Centre Val de Loire
	Yves Le Traon	-	Professeur à l'Université de Luxembourg
Éxaminateurs:	Benoit Baudry	-	Professeur à l'École Royale Polytechnique (KTH)
	Marc Tommasi	-	Professeur à l'Université de Lille
Garant:	Romain Rouvoy	-	Professeur à l'Université de Lille

In loving memory of
Walter Ivan Rudametkin Valoff
1945 - 2021

Thank you for being a great father and raising us in a world full of adventure and knowledge. We'll never forget the surfing, cliff jumping, snorkeling, hang gliding, motorcyles, bow hunting, taming guacamayas, nor the thousands of fishing trips and the countless stories you told to keep them alive and well. But most of all, thanks for keeping your inner child going for over 75 years and thank you for always helping others.

Acknowledgments

Walter Rudametkin
Lille, France
April, 2021

Abstract

I have been an associate professor in computer science at the University of Lille and a member of the Spirals project-team in the CRIStAL laboratory since September 2014. I obtained my PhD in Software Engineering in Grenoble in 2013, focusing on building robust self-adaptive component-based systems, and I completed a postdoctoral stay in the Inria DiverSE project-team, in Rennes, in the area of component-based software engineering. Since 2014, my research has mostly focused on (i) multi-cloud computing and (ii) security and privacy on the web. I have successfully co-advised two doctorates, Gustavo Sousa (defended July 2018) and Antoine Vastel (defended November 2019), and currently advise 3 students. I have decided to write my *Habilitation pour Diriger des Recherches* (HDR) in the area of privacy and security because this will be my main line of research activities for the near future. More specifically, I present the results of the research that my students, colleagues, collaborators, and I have done in the area of *browser fingerprinting*.

Browser fingerprinting is the process of identifying devices by accessing a collection of relatively stable attributes through Web browsers. We call the generated identifiers browser fingerprints. Fingerprints are stateless identifiers and no information is stored on the client's device. In the first half of this manuscript, we identify and study three properties of browser fingerprinting that make it both a risk to privacy, but also of use for security. The first property, *uniqueness*, is the power to uniquely identify a device. We performed a large scale study on fingerprint uniqueness and, although not a perfect identifier, we show its statistical qualities allow uniquely identifying a high percentage of both desktops and mobile devices [Laperdrix 2016]. The second property, *linkability*, is the capacity to re-identify, or link, fingerprints over time. This is arguably the main risk to privacy and enables fingerprint tracking. We show, through two algorithms, that some devices are highly trackable, while other devices' fingerprints are too similar to be tracked over time [Vastel 2018b]. The third and final property is *consistency*, which refers to the capacity to verify the attributes in a fingerprint. Through redundancies, correlations or dependencies, many attributes are verifiable, making them more difficult to spoof convincingly. We show that most countermeasures to browser fingerprinting are identifiable through such inconsistencies [Vastel 2018a], a useful property for security applications.

In the second half of this manuscript, we look at the same properties from a different angle. We create a solution that breaks fingerprint linkability by randomly generating usable browsing platforms that are unique and consistent [Laperdrix 2015]. We also propose an automated testing framework to provide feedback to the developers of browsers and browser extensions to assist them in reducing the uniqueness or their products [Vastel 2018c]. Finally, we look at how fingerprint consistency is exploited in-the-wild to protect websites against automated Web crawlers. We show that fingerprinting is effective and fast to block crawlers, but lacks resiliency when facing a determined adversary [Vastel 2020].

Beyond the results I report in this manuscript, I draw perspectives for exploring browser fingerprinting for multi-factor authentication, with a planned large-scale deployment in the following months. I also believe there is potential in automated testing to improve privacy. And of course, we know that fingerprint tracking does not happen in a bubble, it is complementary to other techniques. I am therefore exploring other tracking techniques, such as our preliminary results around IP addresses [Mishra 2020] and caches [Mishra 2021], using ad blockers against their users, and a few other ideas to improve privacy and security on the Web.

Keywords: Browser fingerprinting, Web tracking, Multi-factor authentication, Security and Privacy

Contents

List of Figures viii

List of Tables x

1 Introduction 1

2 Background 7
2.1 Context and motivations . . . 7
2.2 Browser Fingerprinting . . . 8
2.2.1 Browser fingerprint *uniqueness* . . . 8
2.2.2 Browser fingerprint *linkability* . . . 12
2.2.3 Browser fingerprint *consistency* . . . 12
2.2.4 Browser fingerprinting countermeasures . . . 13
2.2.5 Other Forms of Fingerprinting. . . . 14
2.3 Non-fingerprinting detection methods for crawlers . . . 15
2.4 Summary . . . 15

3 Uniqueness, Linkability and Consistency 17
3.1 Motivations . . . 17
3.2 Overview . . . 18
3.3 Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints . . . 19
3.3.1 Dataset . . . 20
3.3.2 Canvas Fingerprinting . . . 21
3.3.3 Mobile Device Fingerprinting . . . 22
3.3.4 Discussing *Beauty and the Beast* . . . 24
3.4 FP-Stalker : Tracking Browser Fingerprint Evolutions . . . 25
3.4.1 Linking Browser Fingerprints . . . 28
3.4.1.1 Rule-based Linking Algorithm . . . 28
3.4.1.2 Hybrid Linking Algorithm . . . 29
3.4.2 Empirical Evaluation of Fp-Stalker . . . 33
3.4.3 Discussing Fp-Stalker . . . 38
3.5 FP-Scanner: Detecting Browser Fingerprint Inconsistencies . . . 38
3.5.1 Operating System Inconsistencies . . . 39
3.5.2 Browser Inconsistencies . . . 40
3.5.3 Device Inconsistencies . . . 41
3.5.4 Canvas Inconsistencies . . . 41
3.5.5 Empirical Evaluation of FP-Scanner . . . 42
3.5.6 Discussing FP-Scanner . . . 45
3.6 Summary . . . 47

4 Breaking Linkability, Reducing Uniqueness, and Exploiting Consistency 51
4.1 Motivations . . . 51
4.2 Overview . . . 52
4.3 Blink: Breaking *Linkability* Through Multi-Level Reconfiguration . . . 53
4.3.1 A moving target defense against tracking . . . 53
4.3.1.1 Balancing privacy and browsing comfort . . . 55
4.3.1.2 Dissimilarity metric . . . 55
4.3.1.3 Comparing Blink . . . 56

4.3.2 Multi-level reconfiguration . . . 57
4.3.2.1 The diversity reservoir . . . 58
4.3.2.2 Modeling the well-formedness constraints in the diversity reservoir 58
4.3.3 Empirical validation of multi-level reconfiguration . . . 59
4.3.4 Discussing Blink . . . 60
4.4 FP-TESTER: Automated Testing to Reduce Fingerprint Uniqueness . . . 61
4.4.1 Short-Term Fingerprintability of Browser Extensions . . . 61
4.4.2 Long-Term Fingerprintability of Browser Extensions . . . 62
4.4.3 Fingerprintability Reports and Components . . . 63
4.4.4 Preliminary Results: RANDOM AGENT SPOOFER . . . 64
4.4.5 Discussing FP-TESTER . . . 65
4.5 FP-CRAWLERS: exploiting fingerprint consistency to block crawlers . . . 66
4.5.1 Websites that block crawlers with fingerprinting . . . 66
4.5.2 Characterizing fingerprinting scripts . . . 67
4.5.3 Effectiveness of browser fingerprinting to detect crawlers . . . 68
4.5.4 Discussing FP-CRAWLERS . . . 71
4.6 Summary . . . 71

5 Conclusions and Perspectives **75**

Bibliography **81**

List of Figures

2.1 Example of two canvas fingerprints used by commercial fingerprinting scripts. . . . 9
2.2 Presentation of the different attributes related to the size of the screen and the window. . . . 10

3.1 Example of a canvas fingerprint . . . 21
3.2 Comparison of the *Smiling face with open mouth* emoji on different devices. . . . 22
3.3 A *diff* of two canvas fingerprints from devices with different hardware but the same browser and OS (differences in red). . . . 22
3.4 Comparison of anonymity set sizes on the list of plugins between desktop and mobile devices . . . 23
3.5 Comparison of the user agent anonymity set sizes on desktop and mobile devices . 23
3.6 Comparison of anonymity set sizes of user agents from Chrome and Firefox on mobile devices . . . 24
3.7 Comparison of anonymity set sizes of user agents from Android and iOS devices . 25
3.8 Number of fingerprints and distinct browser instances per month . . . 26
3.9 Browser fingerprint anonymity set sizes . . . 26
3.10 CDF of the elapsed time before a fingerprint evolution for all the fingerprints, and averaged per browser instance. . . . 28
3.11 Fp-Stalker: Overview of both algorithm variants. The rule-based algorithm is simpler and faster but the hybrid algorithm leads to better fingerprint linking. . . 29
3.12 First 3 levels of a single tree classifier from our forest. . . . 33
3.13 Overview of our evaluation process that allows testing the algorithms using different simulated collection frequencies. . . . 34
3.14 Example of the process to generate a simulated test set. The dataset contains fingerprints collected from browser's `A` and `B`, which we sample at a $collect\_frequency$ of 2 days to obtain a dataset that allows us to test the impact of $collect\_frequency$ on fingerprint tracking. . . . 35
3.15 Average *tracking duration* against simulated collect frequency for the three algorithms 36
3.16 Average maximum tracking duration against simulated collect frequency for the three algorithms. This shows averages of the longest tracking durations that were constructed. . . . 36
3.17 Average number of assigned ids per browser instance against simulated collect frequency for the three algorithms (lower is better). . . . 37
3.18 Average ownership of tracking chains against simulated collect frequency for the three algorithms. A value of 1 means the tracking chain is constructed perfectly. . 37
3.19 CDF of average and maximum tracking duration for a collect frequency of 7 days (Fp-Stalker hybrid variant only). . . . 37
3.20 Overview of the inconsistency test suite. . . . 39
3.21 Two examples of canvas fingerprints (a) a genuine canvas fingerprint without any countermeasures installed in the browser and (b) a canvas fingerprint altered by the Canvas Defender countermeasure that applies a uniform noise to all the pixels in the canvas. . . . 42

4.1 Browsing platform elements exhibited in the browser fingerprint. . . . 54
4.2 Evolution of the user's platform over time. . . . 55
4.3 A multi-level view of browsing platforms. Virtualization isolates the user's system. 57
4.4 An extract of the feature model used for assembling valid DPC configurations. . . 59
4.5 Dissimilarity between consecutive platforms (Leery mode) . . . 60
4.6 Dissimilarity between consecutive platforms (Coffee break mode) . . . 60

4.7 Overview of FP-TESTER toolkit 61
4.8 Short-term fingerprintability report provided by FP-TESTER 65
4.9 Overview of FP-CRAWLERS. 66
4.10 Crawl efficiency statistics. 70

List of Tables

2.1 An example browser fingerprint. 11

3.1 Browser measurements of AmIUnique fingerprints. 20
3.2 Summary of statistics 21
3.3 Durations the attributes remained constant for the median, the 90^{th} and the 95^{th} percentiles. 27
3.4 Feature importances of the random forest model calculated from the train set. . . . 32
3.5 Number of fingerprints per generated test set after simulating different collect frequencies 36
3.6 Mapping between common `OS` and `platform` values. 40
3.7 Mapping between common `OSes` and WebGL's `renderer`/`vendor` attributes. . . . 40
3.8 List of attributes collected by our fingerprinting script. 42
3.9 Comparison of accuracies per countermeasures 43
3.10 FP-Scanner steps failed by countermeasures 45

4.1 Changed attributes for example n°1 56
4.2 Changed attributes for example n°2 56
4.3 Fingerprinting tests and the scripts that use them. ✓ indicates the attribute is collected and a verification test is run in the script. ∼ indicates the attribute is collected but no tests are run directly in the script. The absence of symbol indicates that the attribute is not collected by the script. 68
4.4 List of crawlers and altered attributes. 69

Chapter 1

Introduction

I have been an associate professor in computer science at the University of Lille and a member of the Spirals project-team in the CRIStAL laboratory since September 2014. I obtained my doctorate in Software Engineering in Grenoble in 2013, focusing on building robust self-adaptive component-based systems, and I completed a postdoctoral stay in the Inria DiverSE project-team, in Rennes, in the area of component-based software engineering and model-driven engineering. Since 2014, my research has mostly focused on *(i)* multi-cloud computing and *(ii)* security and privacy on the web. I have been, arguably, the main co-advisor of two successful doctorates, Gustavo Sousa (defended July 2018) and Antoine Vastel (defended November 2019), and currently co-advise two third-year doctoral students, and, after obtaining a dispensation, I am the sole advisor of a first-year doctoral student. I have also participated in multiple other PhDs, and have supervised over two dozen first and second year master's degree (or equivalent) internships and projects.

I have decided to write my manuscript for the degree of *Habilitation pour Diriger des Recherches (HDR)*, loosely translated to English as *Habilitation to Direct Research*, in the area of privacy and security because this will be my main line of research activities for the near future. More specifically, I present the results of the research that my students, colleagues, collaborators and I have done in the area of browser fingerprinting. The decision to focus on this topic is a practical consideration; the research is recent, I'm co-advising ongoing PhDs on the topic, we have broadly covered the topic with multiple results and contributions of which I am proud of, and it fits together to tell a lovely story, in my humble opinion. This choice is, of course, not intended to reduce the importance nor the quality of our results in other areas.

Browser fingerprinting is the process of identifying devices by accessing a collection of relatively stable attributes through Web browsers. We call the generated identifiers *browser fingerprints*. Fingerprints are stateless identifiers and no information is stored on the client's device. In the first half of this manuscript, we identify and study three properties of browser fingerprinting that make it both a risk to privacy, but also of use for security. The first property, **uniqueness**, is the power to uniquely identify a device. We performed a large scale study on fingerprint uniqueness and, although not a perfect identifier, we show its statistical qualities allow uniquely identifying a high percentage of both desktop computers and, more surprisingly at the time, mobile devices [Laperdrix 2016]. The second property, **linkability**, is the capacity to re-identify, or link, fingerprints from the same device over time. This is arguably the main risk to privacy and enables fingerprint tracking. We show, through two algorithms, that some devices are highly trackable, while other devices' fingerprints are too similar to be tracked over time [Vastel 2018b]. The third and final property is consistency, which refers to the capacity to verify the attributes in a fingerprint. Through redundancies, correlations or dependencies, many attributes are verifiable, making them more difficult to spoof convincingly. We show that most countermeasures to browser fingerprinting are identifiable through such inconsistencies [Vastel 2018a], a useful property for security applications.

In the second half of this manuscript, we look at the same properties from a different angle. We create a solution to **break fingerprint linkability** by randomly generating usable browsing platforms that are unique and consistent [Laperdrix 2015]. We also propose an automated testing framework to provide feedback to the developers of browser fingerprinting countermeasures to assist them in **reducing the uniqueness** or their products [Vastel 2018c]. Finally, we look at how **fingerprint consistency** is exploited in-the-wild to protect websites against automated Web crawlers. We show that fingerprinting is effective and fast to block crawlers, but lacks

resiliency when facing a determined adversary [Vastel 2020].

Beyond the results I report in this manuscript, I draw perspectives for exploring browser fingerprinting for multi-factor authentication, with a planned large-scale deployment in the following months. I'm keen on finding automated [Bird 2020] and semi-automated [Durey 2021] ways to understand the uses and risks of browser fingerprinting, as well was an understanding how representative and precise we are in the manner we conduct our studies [Zeber 2020]. I also believe there is a lot of potential in automated testing to improve privacy. We can and will go beyond *Fp-Tester*. And of course, we know that fingerprint tracking does not happen in a bubble, it is complementary to other techniques. I am therefore exploring other tracking techniques, such as extending our preliminary results around IP addresses [Mishra 2020] and caches [Mishra 2021], using ad blockers and filter lists to track users, and a few other ideas to improve privacy and security on the Web.

Doctoral student supervision

Naif MEHANNA, 2020-2023 (100%, sole advisor)
Turning Ad Blockers Into Trackers.
Financed through the ANR JCJC FP-Locker project.

Antonin DUREY, 2018-2021 (50%, co-advised with Romain Rouvoy & Lionel Seinturier)
Leveraging Browser Fingerprinting to Fight Fraud on the Web.
Financed through a collaboration with the French Army Ministry.

Vikas MISHRA, 2018-2021 (50%, co-advised with Romain Rouvoy & L. Seinturier)
Contributions to tracking techniques : caches, IP addresses, fingerprinting.
Doctoral contract, Inria CORDIS.

Antoine VASTEL, 2016-2019 (66%, co-advised with Romain Rouvoy)
Tracking Versus Security: Investigating the Two Facets of Browser Fingerprinting.
Doctoral contract, Université de Lille
Defended October 2019. Currently working at Datadome in bot detection.

Benjamin DANGLOT, 2016-2019 (20%, co-advised with M. Monperrus & L. Seinturier)
Amplification automatique de tests unitaires pour DevOps.
Financed obtained by M. Monperrus through the H2020 STAMP project.
Defended November 2019, currently research Engineer.

Miguel GONZALEZ, 2016-2017 (50%, co-advised with Martin Monperrus & R. Rouvoy)
Building and Testing Anti-fragile Blockchain Applications.
CONACyT scholarship (Mexican government).
Stopped September 2017 for personal reasons. Currently an engineer at Oracle campus Guadalajara, Mexico.

Gustavo SOUSA, 2014-2018 (60%, co-advised with Laurence Duchien)
A Software Product Line Approach for the Setup and Adaptation of Multi-Cloud Environments.
ERASMUS Mundus scholarship.
Defended July 2018. Currently an assistant professor in Brazil.

I'd also like to make a special mention of Pierre Laperdrix, who, during my post-doc in Rennes, I co-supervised for his master's degree. His motivation and work on that silly little browser fingerprinting project I proposed helped make all of this possible. And although I didn't co-supervise him for his doctorate, I've enjoyed working with him ever since.

Publications

The following is my list of publications, since my postdoctoral stay, presented in reverse chronological order. The majority have taken place either in a supporting role to a doctorate student, for example, during my postdoctorate, or directly in a supervising role since I obtained my position as an associate professor.

Journal publications

2020 Benjamin Danglot, Martin Monperrus, Walter Rudametkin, et Benoit Baudry. **An approach and benchmark to detect behavioral changes of commits in continuous integration**. Empirical Software Engineering, March 5th, 2020. Pages 1573-7616. https://doi.org/10.1007/s10664-019-09794-7
Scimago Q1. Impact Factor 4.457. Google Scholar (Software Systems): #7

2016 Inti Gonzalez-Herrera, Johann Bourcier, Erwan Daubert, Walter Rudametkin, Olivier Barais, François Fouquet, Jean-Marc Jézéquel and Benoit Baudry. **ScapeGoat: Spotting abnormal resource usage in component-based reconfigurable software systems**. Journal of Systems and Software (JSS) Volume 122, December 2016, Pages 398-415. https://hal.inria.fr/hal-01354999
Scimago Q1. Impact Factor 2.444. Google Scholar (Software Systems): #3

Peer-reviewed conference publications

2021 Antonin Durey, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **FP-Redemption: Studying Browser Fingerprinting Adoption for the Sake of Web Security.** The 18th Conference on Detection of Intrusions and Malware & Vulnerability Assessment - DIMVA'21, Jul 2021, Lisboa, Portugal. https://hal.inria.fr/hal-03212726

2021 Vishra Mishra, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. **Déjà vu: Abusing Browser Cache Headers to Identify and Track Online Users.** PETS 2021 - The 21th International Symposium on Privacy Enhancing Technologies, July 2021, Virtual conference. https://hal.inria.fr/hal-03017222
Core Rank: B / Google Scholar (Computer Security): #15 Acceptance rate: 18%

2020 Vikas Mishra, Pierre Laperdrix, Antoine Vastel, Walter Rudametkin, Romain Rouvoy, Martin Lopatka. **Don't count me out: On the relevance of IP addresses in the tracking ecosystem.** The Web Conference (TheWebConf'20, formerly WWW), Apr 2020, Taipei, Taiwan. https://hal.archives-ouvertes.fr/hal-02456195
Core Rank: A / Google Scholar (Databases & Information systems): #3 – Acceptance rate: 19%*

2020 David Zeber, Sarah Bird, Camila Oliveira, Walter Rudametkin, Ilana Segall, Fredrik Wollsén, Martin Lopatka. **The Representativeness of Automated Web Crawls as a Surrogate for Human Browsing.** The Web Conference (TheWebConf'20, formerly WWW), Apr 2020, Taipei, Taiwan. https://hal.archives-ouvertes.fr/hal-02456195
Core Rank: A / Google Scholar (Databases & Information systems): #3 – Acceptance rate: 19%*

2018 Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **FP-STALKER: Tracking Browser Fingerprint Evolutions.** 39th IEEE Symposium on Security and Privacy (S&P 2018). https://hal.inria.fr/hal-01652021
Core Rank: A / Google Scholar (Computer Security): #2 – Acceptance rate: 11%.*

2018 Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies.** 27th USENIX Security Symposium (USENIX Sec. 2018, Baltimore, USA). https://hal.inria.fr/hal-01820197
Core Rank: A/ Google Scholar (Computer Security): #4 – Acceptance rate: 19%.*

2017 Gustavo Sousa, Walter Rudametkin, Laurence Duchien. **Extending Dynamic Software Product Lines with Temporal Constraints.** 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2017), May 2017, Buenos Aires, Argentina. Pages 129-139. https://hal.inria.fr/hal-01482014

2016 Pierre Laperdrix, Walter Rudametkin, Benoit Baudry. **Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints.** 37th IEEE Symposium on Security and Privacy (S&P 2016). *Rank: A* / Google Scholar (Computer Security): #2 – Acceptance rate: 13%.*
⧗ **2018 CNIL-Inria European Privacy Protection Award.**

2016 Gustavo Sousa, Walter Rudametkin, Laurence Duchien. **Extending Feature Models with Relative Cardinalities.** Systems and Software Product Line Conference (SPLC'16), Sep 2016, Beijing, China. 2016. Pages 79-88. https://hal.inria.fr/hal-01312751

2016 Gustavo Sousa, Walter Rudametkin, Laurence Duchien. **Automated Setup of Multi-Cloud Environments for Microservices-Based Applications.** 9th IEEE International Conference on Cloud Computing (CLOUD), Jun 2016, San Francisco, USA. Pages 327-334. https://hal.inria.fr/hal-01312606 *Core Rank: B – Acceptance rate: 15%.*

2016 Inti Gonzalez-Herrera, Johann Bourcier, Walter Rudametkin, Olivier Barais, Francois Fouquet. **Squirrel: Architecture Driven Resource Management**. SAC - 31st Annual ACM Symposium on Applied Computing (SAC'16), Apr 2016, Pisa, Italy. Pages 1329-1336. https://hal.inria.fr/hal-01355000 *Core Rank: B*

2015 Pierre Laperdrix, Walter Rudametkin and Benoit Baudry. **Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification.** Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'15), Florence, Italy, May 2015. Pages 98-108. https://hal.inria.fr/hal-01121108 *Acceptance rate: 29%.*

2014 Inti Gonzalez-Herrera, Johann Bourcier, Erwan Daubert, Walter Rudametkin, Olivier Barais, François Fouquet, Jean-Marc Jezequel. **Scapegoat: an Adaptive monitoring framework for Component-based systems.** IEEE/IFIP Conf. on Software Architecture (WICSA 2014), Sydney, Australia. https://hal.inria.fr/hal-00983045 *Core rank A, Acceptance rate 21.1%.*

Peer-reviewed workshop publications

2020 Antoine Vastel, Walter Rudametkin, Romain Rouvoy, Xavier Blanc. **FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers.** MADWeb'20 - NDSS Workshop on Measurements, Attacks, and Defenses for the Web, Feb 2020, San Diego, United States. https://hal.archives-ouvertes.fr/hal-02441653.
⧗ **Best Paper Award.**

2018 Antoine Vastel, Walter Rudametkin, Romain Rouvoy. **FP-TESTER: Automated Testing of Browser Fingerprint Resilience.** 4th International Workshop on Privacy Engineering IWPE 2018, April 2018. https://hal.inria.fr/hal-01717158

2017 Miguel Alejandro González, Walter Rudametkin, Martin Monperrus, Romain Rouvoy. **Challenging Anti-fragile Blockchain Applications.** 11th EuroSys Doctoral Workshop EuroDW'17, Apr 2017, Belgrade, Serbia. https://hal.inria.fr/hal-01653986

2016 Gustavo Sousa, Walter Rudametkin, Laurence Duchien. **Software Product Lines for Multi-Cloud Microservices-Based Applications.** 6th International Workshop on Cloud Data and Platforms (CloudDP), Apr 2016, London, United Kingdom. https://hal.inria.fr/hal-01302184

2014 Gustavo Sousa, Walter Rudametkin, Laurence Duchien. **Challenges for Automatic Multi-Cloud Configuration.** JLDP 14 - Journée Lignes de Produits, Dec 2014, Luxembourg, Dec 2014, Luxembourg, Luxembourg. https://hal.inria.fr/hal-01093220

Non peer-reviewed articles

2021 Antonin Durey, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **An iterative technique to identify browser fingerprinting scripts. 2021.** https://hal.inria.fr/hal-03212729

2020 Sarah Bird and Vikas Mishra and Steven Englehardt and Rob Willoughby and David Zeber and Walter Rudametkin and Martin Lopatka, **Actions speak louder than words: Semi-supervised learning for browser fingerprinting detection**, 2020. https://arxiv.org/abs/2003.04463

Research grants

2019 **ANR JCJC FP-Locker : Hardening web authentication with browser fingerprinting** — Leader. The project's main objective is to develop an authentication system that uses browser fingerprinting to protect websites against fraud and bots. The project started in September 2020 with the recruitment of Naif Mehanna and includes members from the the CNRS and Mozilla. Ends 2024. Total funding: 168k€

2018 Research and collaboration contract with *Ministère des Armées* on the topic of **Leveraging Browser Fingerprinting to Fight Fraud on the Web**. Antonin Durey is funded under this contract. Total funding: 163k€.

2018 Inria ADT (*Action de Développement Technologique*) funding. The **FingerKit: a Cloud Platform to Study Browser Fingerprints at Large** project served to finance 24 engineer-months on a full rebuild of the AmIUnique platform with new functionality.

2018 Inria CORDI-S doctoral grant on **Collaborative Strategies to Protect from Browser Fingerprinting**. This grant finances Vikas Mishra's doctoral contract.

2016 **PEPS JCJC INS2I FPDefendor**. Project to explore techniques to protect users against browser fingerprint tracking. Total funding: 8700€.

2016 Université de Lille doctoral grant on the subject of **Machine learning to optimize privacy against browser fingerprint tracking**. This grant financed Antoine Vastel's doctoral contract.

Structure of this document

The rest of this document is structured as follows:

In **Chapter 2 Background** I present the initial motivations and the context of our work, as well as the background and related work.

Chapter 3 Uniqueness, Linkability and Consistency describes the three main properties of browser fingerprinting that we have identified. These properties make it a risk to privacy but also make it useful for security. We explore these properties through three studies.

Chapter 4 Breaking Linkability, Reducing Uniqueness, and Exploiting Consistency explores the defenses and the uses of browser fingerprinting. Through three studies, we show that we can break fingerprint tracking using random reconfigurations and that we can assist developers in reducing the fingerprintability of their products. We also show how fingerprinting can be exploited in order to detect crawlers and how resilient it is to adversaries.

Finally, I give my final remarks in **Chapter 5 Conclusions and Perspectives**.

Chapter 2

Background

Contents

2.1 Context and motivations 7
2.2 Browser Fingerprinting 8
2.2.1 Browser fingerprint *uniqueness* 8
2.2.2 Browser fingerprint *linkability* 12
2.2.3 Browser fingerprint *consistency* 12
2.2.4 Browser fingerprinting countermeasures 13
2.2.5 Other Forms of Fingerprinting. 14
2.3 Non-fingerprinting detection methods for crawlers 15
2.4 Summary 15

2.1 Context and motivations

The Web is an essential part of our daily lives. If your reading this manuscript, an important part of your life probably revolves around the amazing qualities and benefits it has brought. A priceless tool for communicating that has brought us closer than ever. However, the Web is also a place where everything we do, say, read, watch can be recorded, dissected, studied, monetized and used to influence us. Today, most of the Web collects this information through Ad networks, and many argue that this is necessary to keep services free and open. My work isn't to argue about the ethics of this—of course I have my own opinion—but to to talk about the techniques that are used in identifying, and more specifically, re-identifying devices. Many of us now have personal devices, meaning there's a one-to-one mapping between my device and myself. By tracking, collecting, analyzing the data of my device, the Web is in essence, dissecting my persona.

Cookies are the traditional and arguably most effective way the Web has to re-identify devices. If, in the real world, you went into a store and they gave you, and only you, a unique number, and everywhere you go you show this number to any and everybody you see, you might begin to worry about your privacy. And if you did this for every store, street, company or place you visited, and shared your unique numbers all the time? And if you let the store's partners also give you unique numbers and ask for them back later? The implications are extensive and the discussions endless. And in an overly simplified manner, this is what websites and ad networks do with cookies, both first-party or third-party. A website asks you to store a unique ID, and anytime you come back, you return it saying "hey, it's me again". Then websites and Web services sync their unique IDs, and resync them, and re-resync them, as you travel across the Web, re-identifying you and everything you do. This allows them to not only follow what you did on one website, but indeed, to monitor what you do across a large multitude of websites. In the real world, this is known as stalking, but on the Web, this is just business.

However, legislation, such as General Data Protection Regulation (GDPR) or California's Consumer Privacy Act (CCPA), has brought to light the privacy concerns that these cookies imply; more people are sensitive to these issues. A study conducted by Microsoft in 2012 observed that they were unable to keep track of 32 % of their users using only cookies, as they were regularly deleted [Yen 2012]. Cookie erasure is now common, third-party cookies are often blocked, and many browsers and browser extensions let users manage their cookies. The use of private browsing

modes, which automatically delete cookies at the end of browsing sessions, is common. Some browsers even specialize in protecting the user's privacy. But as is often the case in these matters, there has been a search for a replacement to cookies. The ideal solution being sought by trackers should not depend on being stored in the user's device, should consistently re-identify them, and it would be even better if this process was hard for user's to see or check for. Just to give an example that this battle is ongoing, Google has a brand new and highly criticized browser standard, called FloC, that would group internet users into *cohorts* and provide group-wide advertising IDs, potentially introducing a new vector for tracking [Foundation 2021a] that would replace third-party cookies.

It should be noted that at different times, the Web has seen moments of rapid growth in technology, increasing the diversity of APIs, and creating strong incentives to produce features faster and outperform the competition [1]. Recently, Google Chrome has taken a strong lead and is the browser of choice for about 2/3rds of the Web [2]. This has led to, for the first time, some browsers attempting to differentiate themselves through privacy features and even browser fingerprinting protection, such as *Intelligent Tracking Prevention* in Safari, *Enhanced Tracking Protection* in Firefox, or *Shields* in Brave.

In this manuscript, we focus on **browser fingerprinting**, with is an identification and re-identification technique, using information gained from the standard HTTP and JavaScript APIs to create unique identifiers. Browser fingerprinting depends on the collection and comparison of browsers' attributes to build unique identifiers and has been the main focus of our research for the last 7 years. It has stirred quite a commotion and is often exaggerated: by some who claim it has an extraordinary capability to uniquely identify and track devices, and by others who claim it is an overly ephemeral identifier that is not useful at all. Our research shows that a browser fingerprint is an imperfect identifier [Laperdrix 2016, Vastel 2018b] yet can still be useful [Vastel 2020] and difficult to avoid [Laperdrix 2015, Vastel 2018a, Vastel 2018c]. Although it can't uniquely re-identify every device, some are just too common or the attributes too unstable, it can consistently re-identify many devices, in particular those that are overly customized or rare. I argue that browser fingerprinting is useful for re-identifying devices and respawning cookies that have been deleted, and allows tracking many devices that have taken measures to protect themselves from cookie-based tracking. It is likely that these people are more technically literate, and their devices more customized, meaning they are more identifiable through fingerprinting, and thus are ideal targets for this type of attack. Cookie respawning through browser fingerprinting has also been shown in the wild [Acar 2014]. If you don't customize your browser at all, your browser is victim to cookie-based tracking, but if you over customize and properly delete your cookies, you're a candidate for fingerprint-based tracking. In the rest of this chapter I go over the background and related work to browser fingerprinting.

We have studied the three main properties of browser fingerprints, namely their *uniqueness*, their *linkability*, and their *consistency* and verifiability. The rest of this chapter present related work and background to browser fingerprinting. We have studied are particularly interested in its three main properties, *uniqueness*, *linkability* and consistency.

2.2 Browser Fingerprinting

2.2.1 Browser fingerprint *uniqueness*

In his master's thesis, Mayer [Mayer 2009] showed that browsers could be uniquely identified because of their configuration. To adapt a website to the user's device, browsers query the device's configuration using JavaScript APIs. The combination of attributes collected from these APIs can be collected without consent and used to create a unique identifier that we now call a *browser fingerprint*. In 2010, Eckersley introduced a tracking technique called *browser fingerprinting* [Eckersley 2010] in the first large-scale study of browser fingerprinting, in particular,

[1] https://en.wikipedia.org/wiki/Browser_wars
[2] https://gs.statcounter.com/browser-market-share#monthly-202102-202103

on fingerprint *uniqueness*. With the Electronic Frontier Foundation, he created the Panopticlick website[3] and collected more 470K browser fingerprints, among which 83.6% could be uniquely identified from a fingerprint composed of only 8 attributes. He also showed that more than 94.2% of the fingerprints when Flash or Java plugins were activated (thanks to the list of fonts). Because of this uniqueness, he argued browser fingerprints can be used for tracking.

Following Eckersley's study, several others have measured the use of fingerprinting in the wild [Nikiforakis 2013, Acar 2014, Englehardt 2016]. They all showed fingerprinting being used by a significant fraction of popular websites. We could see commercial fingerprinters adapting their behavior to leverage new APIs. For example, in 2013, Nikiforakis et al. [Nikiforakis 2013] showed that none of the commercial fingerprinters were still using Java, and Englehardt et al. [Englehardt 2016] showed that fingerprinters had found new approaches to exploit APIs introduced by HTML5.Further studies have focused on studying new attributes that increase browser fingerprint uniqueness [Cao 2017, Fifield 2015, Mowery 2011, Mowery 2012, Mulazzani 2013]. And although there has been some effort by vendors to address these issues, the just shifts. As browsers have grown and taken on the role of traditional native applications, with new APIs and hardware acceleration, their attack surfaces have also grown, allowing many more attributes to be used for re-identification. Even simple things, like the order of the HTTP headers [Yen 2012], or over detailed information about the devices battery can be used to fingerprint [Olejnik 2016].

Mowery et al. [Mowery 2012] showed that the HTML canvas API could be used to generate images whose rendering depends on the browser and the device. These canvas use different techniques that, when combined, generate an image whose rendering is highly unique. For example, Acar et al. [Acar 2014] showed that commercial fingerprinters used strings that are pangrams—i.e., strings constituted of all the letters of the alphabet—or use emojis since their rendering depends on the OS and the kinds of device. Figure 2.1 presents the canvas generated by Akamai and PerimeterX fingerprinting scripts. The browser also exposes different properties, through the `screen` and the `window` objects, that reflect the size of the screen and the window. Figure 2.2 presents a screenshot of a browser on MacOS that shows how these attributes relate to each other. One can imagine how these could be cross-checked to verify they are consistent.

(a) Canvas fingerprint generated by Akamai Bot Manager fingerprinting script.

(b) Canvas fingerprint generated by PerimeterX fingerprinting script.

Figure 2.1: Example of two canvas fingerprints used by commercial fingerprinting scripts.

An example browser fingerprint is presented in Table 2.1. While each attribute need not be unique, the objective is to find a combination of attributes, i.e., a browser fingerprint, that

[3]Was `https://panopticlick.eff.org`, now `https://coveryourtracks.eff.org`

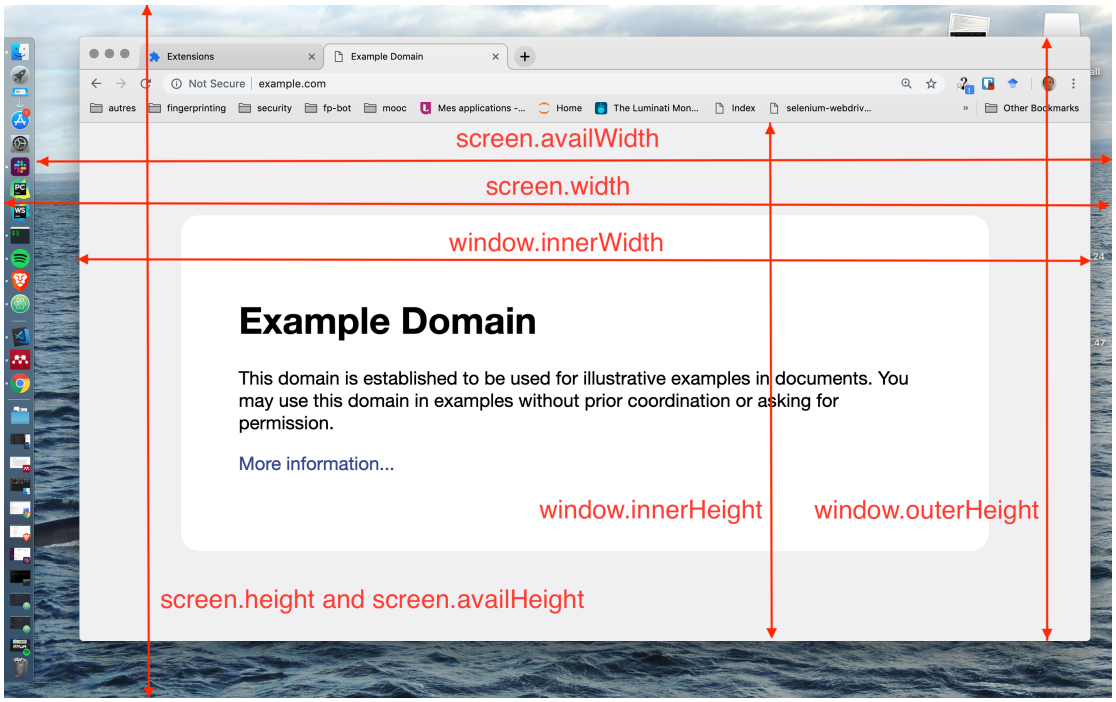


Figure 2.2: Presentation of the different attributes related to the size of the screen and the window.

is unique and stable. These attributes are collected through either HTTP headers or through JavaScript. Typically the IP address or geolocation are not considered part of the browser fingerprint, and neither are other techniques such as TCP fingerprinting [Zalewski 2019].

There's been no shortage of techniques to increase uniqueness. Mowery et al. [Mowery 2011] used the SunSpider and the V8 benchmarks to build a fingerprint, while Sanchez et al. proposed measuring the time to execute sequences of cryptographic functions to generate fingerprints capable of distinguishing [Sanchez-Rola 2018] (interestingly, we were not able to reproduce their results). Browser extensions can also be detected through bugs or side effects. Mowery et al. [Mowery 2011] infer the list of websites whitelisted by the NoScript extension by observing whether from a domain were blocked. Starov et al. [Starov 2017] showed that browser extensions could be identified because of the way they interact with the DOM. Among the 10,000 most popular extensions of the Chrome store, around 15% had a unique way to interact with the DOM, making their presence detectable. They also showed that among 854 users, 14.1% had a unique set of browser extensions. Sjosten et al. [Sjösten 2017] leveraged Web Accessible Resources (WAR) to test the presence of browser extensions. Their approach was able to detect more than 50% of the top 1,000 Chrome extensions. Even though Firefox protected against this kind of attacks by randomizing each extension identifier,[4] Sjosten et al. [Sjösten 2018] showed it was still possible to test the presence of extensions by tricking them to reveal themselves.

Regarding large-scale studies of fingerprint uniqueness, there have been few. After Eckersley's study [Eckersley 2010], and our study, *Beauty and the Beast*, which we present in Section 3.3, there's been one major study. Between 2016 and 2017, Gomez et al. [Gómez-Boix 2018] collected more than 2 million fingerprints on a popular french website from the Alexa's Top 15. Since it is

[4]Protecting against extension probing: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/web_accessible_resources

Table 2.1: An example browser fingerprint.

Attribute	Source	Value Examples
`User-agent`	HTTP header	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36
`Accept`	HTTP header	text/html,application/xhtml+xml,application /xml;q=0.9,image/webp,*/*;q=0.8
`Encoding`	HTTP header	gzip, deflate, sdch, br
`Languages`	HTTP header	en-US,en;q=0.8,es;q=0.6
`Plugins`	JavaScript	Plugin 0: Chrome PDF Viewer; ; mhiehjai. Plugin 1: Chrome PDF Viewer; Portable Document Format; internal-pdf-viewer. Plugin 2: Native Client; ; internal-nacl-plugin.
`Cookies`	JavaScript	yes
`Local storage`	JavaScript	no
`Timezone`	JavaScript	-180
`Resolution`	JavaScript	2560x1440x24
`Fonts`	Flash	Abyssinica SIL,Aharoni CLM,AR PL UMing CN,AR PL UMing HK, ...
`Headers`	HTTP header	Connection Accept X-Real-IP DNT Cookie Accept-Language Accept-Encoding User-Agent Host
`Platform`	JavaScript	Linux x86_64
`Do not track`	JavaScript	yes
`Canvas`	JavaScript	Cwm fjordbank glyphs vext quiz, 😃 Cwm fjordbank glyphs vext quiz, 😃
`WebGL Vendor`	Javascript	NVIDIA Corporation
`WebGL Renderer`	Javascript	NVIDIA GeForce GTX 750 Series
`Ad blocker`	JavaScript	no

a popular website with a wide range of users, it avoids the biases of the Eckersley and Laperdrix studies, namely attracting more customized browser configurations from more technical users. Gomez et al. compared the diversity of fingerprints in their dataset to the ones from Eckersley and to ours. Despite collecting the same set of attributes, with the exception of the canvas that was modified to obtain a higher uniqueness, they found significantly less uniqueness, only 33.6% in their dataset. The difference is even more important for mobile devices. While 81% of the mobile fingerprints collected on AmIUnique were unique, only 18.5% of the fingerprints in their dataset are unique, despite showing the same attributes as having the most entropy (e.g., canvas and user agent), their study showed much less entropy overall.

Their study probably shows that browser fingerprint uniqueness has been overestimated in the past, maybe due to small or biased datasets, their study presents defects of its own. To compare fingerprint the entropy of attributes and fingerprint uniqueness, they restricted their study to the same 17 attributes already collected by Eckersley [Eckersley 2010] and our study [Laperdrix 2016], and did not consider several new attributes available at the time. Thus, the main critic of this study is that it While this study properly evaluates the attributes collected, it underestimates fingerprint uniqueness by excluding newer attributes. In particular, they did not consider the following `navigator.enumerateDevices`, *audio fingerprinting* [Englehardt 2016] (for which Englehardt et al. estimated an entropy of 5.4 bits), `window.innerHeight/Width` or `window.outerHeight/Width`, *audio and video codecs* such as `HTMLMediaElement.canPlayType`, *touch screen APIs* like `navigator.maxTouchPoints`, or `navigator.hardwareConcurrency`.[5] It is ultimately unclear how different the fingerprint uniqueness would have been had included these

[5] https://developer.mozilla.org/en-US/docs/Web/API/NavigatorConcurrentHardware/hardwareConcurrency

attributes, especially since in their study they show that if the user makes any small change to their fingerprint, they quickly become unique.

2.2.2 Browser fingerprint *linkability*

There's been surprisingly little work published on the process of *linking* fingerprints over time. Eckersley [Eckersley 2010] argues that browser fingerprints can be used for tracking, in particular as a mechanism to regenerate supercookies or deleted cookies, because they are unlikely to drastically change over time. To support this claim, they proposed a simple heuristic that aims at linking fingerprints from the same. First, they studied the stability of browser fingerprints over time and showed that among the 8,833 users that had accepted a cookie and that had visited the websites multiple times, more than 37% displayed at least one change in their fingerprint. Nevertheless, they are aware this number may be overestimated because their website attracts people that change their fingerprint on purpose to test. Nevertheless, despite these frequent changes in the fingerprint, his naive heuristic was able to make correct predictions 65% of the time, incorrect predictions 0.56% of the times. Otherwise, 35% of the time, it made no prediction.

In Section 3.4 we show that fingerprints change rather quickly, making it initially unclear to what extent fingerprints from the same device can be linked over time. We present our approach to linking fingerprints through two algorithms: a rule-based algorithm that is fast, and a hybrid algorithm that uses machine learning to increase accuracy. We find that some devices are too similar to track over meaningful periods of time, while others are very trackable.

Although few studies have addressed linkability, various countermeasures have been proposed to do the opposite, break linkability to avoid tracking, as we present below.

2.2.3 Browser fingerprint *consistency*

Niforakis [Nikiforakis 2013] and Acar [Acar 2013] evaluated the effectiveness of fingerprinting countermeasures, such as simple user agent spoofers, or Fireglove, a browser extension that randomly lies about attributes constituting a fingerprint. Their evaluations showed that countermeasures could be detected because they generate inconsistent fingerprints. Thus, they argued that using these kinds of countermeasures could be counterproductive for a user since she could become identifiable. Starov *et al.* [Starov 2017] looked at the notion of *browser extension fingerprintability* (*i.e.*, identifying installed browser extensions). To identify extensions, they observe changes to the DOM and extract a signature. Additionally, Acker *et al.* [Acker 2017] showed that, for Google Chrome, it is possible to identify extensions using public resources.

The main use-case for fingerprint consistency in the literature is to enhance authentication [Unger 2013, Alaca 2016, Van Goethem 2016, Preuveneers 2015] by using the fingerprint as a second factor. Burztein et al. [Bursztein 2016] showed that browser fingerprinting can also be used to detect crawlers. They rely on canvas fingerprinting to create dynamic challenges to detect emulated or spoofed devices used to post fake reviews on the App Store. Contrary to techniques that uniquely identify a device, they verify the devices class by drawing geometric shapes and texts that are expected to render similarly on devices of the same class. They proposed a dynamic challenge-response protocol that leverages the unpredictability and yet stable nature of canvas rendering to detect devices that lie about their nature (e.g., emulated devices or devices that modify the browser and OS contained in their user agent). Laperdrix et al. [Laperdrix 2019] use a similar technique to generate dynamic canvas tests but their solution doesn't depend on massive datasets. Recent studies [Jueckstock 2019, Jonker 2019] show that websites and bot detection companies heavily rely on the presence of attributes added by instrumentation frameworks and headless browsers, such as `navigator.webdriver`, to detect crawlers.

More recently, in 2019, Schwarz et al. [Schwarz 2019] proposed an approach to automatically learn the browser fingerprint differences between browsers. While their approach can be used for targeting exploits that work only on specific OS, architecture or browser, it can also be used to detect the presence of privacy-enhancing extensions. They applied their automated approach to 6 privacy extensions and were able to detect all of them. In particular, they were able to detect

the presence of the Canvas Defender countermeasures because of its side-effects. This continues to show that, paradoxically, privacy-enhancing extensions can be counterproductive since their presence can be detected and increases the identifying information from the browser.

2.2.4 Browser fingerprinting countermeasures

Browser fingerprinting countermeasures fall into three main strategies: blocking fingerprinting scripts, breaking fingerprint linkability, and reducing fingerprint uniqueness.

Blocking fingerprint scripts. Blocking the execution of the scripts makes fingerprinting unusable. While the server can still collect the HTTP headers if the request is made, not collecting JavaScript attributes drastically decreases uniqueness. Extensions that block scripts are among the most popular privacy-enhancing technologies [Pujol 2015]; in March 2019, four of the top ten most popular browser extensions for Firefox where ad-blockers and tracker-blockers [Firefox 2019], such as AdblockPlus [GmbH 2018], uBlock origin [Hill 2018] or Adblock [AdBlock 2018]. These script blocking countermeasures rely, mostly, on crowdsourced filter lists, such as EasyList [EasyList 2018] and EasyPrivacy [EasyPrivacy 2018]. Lists need to be manually updated and require a significant maintenance [Vastel 2018d]. Privacy Badger [Foundation 2021b], on the other hand, uses heuristics to determine if a request should be blocked. When it observes a suspicious third-party on more than three domains, Privacy Badger automatically blocks the content. Merzdovnik et al. [Merzdovnik 2017] quantified the effectiveness of ad-blockers and trackers blockers at scale and show that the majority of blocking tools are effective against stateful third-party trackers, they fail to block well-known stateless trackers that use browser fingerprinting. Englehardt et al. [Englehardt 2016] also showed that popular filter lists tend to detect only a fraction of fingerprinting scripts.

A more radical approach is to block the execution of JavaScript code. The most popular tool for blocking JavaScript is the NoScript [NoScript 2021] browser extension. Other extensions, such as uBlock Origin [Hill 2018] and uMatrix [Hill 2019], as well as browsers such as Brave [Inc. 2018] or, more famously, the Tor browser [Project 2020] also propose convenient mechanisms to disable JavaScript execution. Although effective, many websites become unusable and no longer support dynamic interactions once JavaScript is deactivated. Whitelisting allows users to re-activate scripts that provide important functionality, but this is time consuming, fragile and, from the user's perspective, requires guessing which scripts are needed for what and reloading a page multiple times until it works. As someone who blocks JavaScript extensively, I have failed to activate the right scripts during the particularly sensitive process of payment verification multiple times, I can attest to how annoying this can be and the risks of being blocked and having to contact your bank to reactivate your cards. Interestingly, as shown by Yu et al. [Yu 2016], breaking websites can also lead to users simply disabling their countermeasures.

Breaking fingerprint linkability. Frequently modifying attributes breaks the stability required to track fingerprints over time. A wide range of user agent spoofer extensions change the user agent sent by the browser. The main drawback of spoofers is that they create inconsistent fingerprints [Nikiforakis 2013] (i.e., combinations of attributes that cannot be found in the wild). More advanced extensions, such as Random Agent Spoofer [dillbyrne 2019] use profiles to create more consistent fingerprints but still don't escape this issue since attributes can be cross-verified using various APIs.

PriVaricator [Nikiforakis 2015] is a modified Chrome browser designed specifically to break fingerprint stability through randomization policies so the browser can tell little lies about key fingerprintable features, like the list of plugins or font sizes. PriVaricator succeeds at fooling some well-known fingerprinters while minimizing site breakage. It is limited to a few attributes that, at the time, were considered important. As an anecdote, Nikiforakis implemented and published PriVaricator a while after we presented our ideas for *Blink* (see Section 4.3), but unfortunately our paper got rejected a few time.

Torres et al. [Torres 2015] proposed FP-Block, a browser extension for Firefox that presents a different fingerprint per-site. Their approach focuses on properties of the `navigator` and the `screen` objects. It also adds random noise to canvas fingerprints. FP-Block tries to ensure fingerprint consistency. To do so, they model how different attributes of a fingerprint relate to each other using a Markov chain model.

FaizKhademi et al. [FaizKhademi 2015] proposed FPGuard, a combination of a modified Chromium and a browser extension. The extension detects fingerprinting scripts and blacklists them, while the modified Chromium spoofs attributes. To detect if a script uses fingerprinting they rely on 9 metrics, such as the number of `navigator` and `screen` properties accessed and assign a score that represents a level of suspicion that the script is fingerprinting. They aim at generating fingerprints that "represent the properties of the browser almost correctly" to avoid breakage. FPGuard's authors are aware that their countermeasure can be detected but argue that since the original values cannot be recovered, it is still a privacy improvement.

FPRandom [Laperdrix 2017] is a modified version of Firefox that adds randomness to the canvas and audio APIs since they have high entropy [Laperdrix 2016, Gómez-Boix 2018] and can be modified with little perception.. Contrary to canvas poisoning extensions that apply random noise independently for each pixel, FPRandom Indeed, they modify modifies the `parseColor` function so that whenever a color is added to the canvas, it is slightly modified. Thus, with their approach, all the shapes of the canvas that use a given color will have the same color.

Canvas Defender [Multilogin 2018] is a browser extension available that adds a uniform noise to a canvas. When Canvas Defender installed, it generates four random numbers corresponding to the noise that will be applied to the red, green, blue and alpha components of each pixel. Thus, since the four random noise numbers are constant, the extension is not vulnerable to replay attacks. Nevertheless, as we show in chapter 3.5, Canvas Defender can degrade privacy and the noise can be extracted and used for re-identification.

Reducing fingerprint uniqueness. This strategy aims at increasing the anonymity set of each user. To achieve this goal, one can either block access to attributes or spoof values so that multiple users return the same value. Started from Firefox v41[6], they implement anti-browser fingerprinting features similar to those available in the Tor Browser. Firefox standardizes attributes, such as the user agent, timezone or the number of cores in the CPU. Firefox also blocks access to critical functions used for canvas fingerprinting. Whenever a script tries to access a canvas value using `toDataURL` or `getImageData`, Firefox asks the user for permission[7]. Firefox also blocks several APIs, such as `WEBGL_debug_renderer_info`[8] geolocation and device sensors.

Brave browser [Inc. 2018] is a privacy oriented Chromium-based browser. They block high entropy attributes, such as audio, canvas, and WebGL. Moreover, they also integrates an ad and tracker blocker and a one-click JavaScript (un-)blocking button.

Canvas Blocker [kkapsner 2017] is a Firefox extension that blocks access to the HTML 5 canvas API. Besides blocking, it can also randomize canvas values every time it is retrieved.

2.2.5 Other Forms of Fingerprinting.

While this chapter has focused on browser fingerprinting—*i.e.* the application layer—other forms operate at the network and protocol layers. These forms only identify classes of devices or clients, but are not immune to spoofing [Smart 2000, Frolov 2019]. TCP fingerprinting relies on the IPv4, IPv6 and TCP headers to identify the OS and software sending a request [Anderson 2017]. Akamai analyzed more than 10 million HTTP/2 connections and showed that implementations of the protocol show variances and discrepancies between the user agent and the TCP fingerprint and HTTP/2 fingerprints can reveal the presence of proxies and VPNs.[9] The TLS protocol can

[6] `https://bugzilla.mozilla.org/show_bug.cgi?id=418986`
[7] `https://bugzilla.mozilla.org/show_bug.cgi?id=967895`
[8] `https://bugzilla.mozilla.org/show_bug.cgi?id=1337157`
[9] `https://blogs.akamai.com/2017/06/passive-http2-client-fingerprinting-white-paper.html`

also be used to fingerprint a client [Brotherson 2015] due to differences in the sets of cipher suites and elliptic curves in the client's TLS implementation.

2.3 Non-fingerprinting detection methods for crawlers

A majority of the web's traffic is due to crawlers [Incapsula 2017, networks 2018], which are programs that explore websites to extract data. While some crawlers provide benefits to the websites they crawl (*e.g.*, by increasing visibility from search engines), others scrape valuable data to obtain a competitive advantage. Some businesses crawl their competitors' websites to adjust their pricing strategy, while others copy, republish and monetize content without permission. The legal and moral issues of crawling have been discussed [Bernard 2018, Quora 2018], and companies have sued [Commons 2016] and won against crawlers [Wikipedia 2013].

To protect from undesired crawling, most websites host a `robots.txt` file that specifies the pages that can be crawled and indexed. However, there is no mechanism to force malicious crawlers to respect it. CAPTCHAs [Von Ahn 2003] are popular to detect malicious crawlers, but progress in automatic image and audio recognition, as well as for-profit crowdsourcing services [Sivakorn 2016, 2Captcha 2018, CAPTCHA 2018], mean they can be bypassed [Bock 2017]. Other techniques rely on analyzing the sequence of requests sent by the client. Rate limiting techniques [Stassopoulou 2009, Stevanovic 2012, Balla 2011, Wang 2013] analyze features, such as the number of requests or pages loaded, to classify the client as human or crawler, while more advanced techniques extract features from time series [Jacob 2009]. Chu *et al.* [Chu 2013] leverage behavioral biometrics to detect blog spam bots. Their hypothesis is that humans need their mouse to navigate and their keyboards to type. They collect events, such as keystrokes, and train a decision tree to predict if a user is human. Finally, the use of IP address lists of cloud providers, proxies and VPNs [Zhang 2013], or the timing of operations to detect browsers in virtual machines [Ho 2014], can also be used to indicate crawlers.

Browser fingerprinting is a less studied approach to detect crawlers. Browser fingerprinting addresses some of the weaknesses of state-of-the-art crawler detection techniques, such as, unlike CAPTCHAs, it does not require user interaction, and unlike time series, it can make a decision in a single request. We explore the use of fingerprinting for crawlers in more detail in Section 3.5.

2.4 Summary

We have broken down the three main properties of browser fingerprinting, namely *uniqueness*, *linkability* and *consistency*, and presented the main works for each. These properties represent browser fingerprinting's ability to uniquely identify devices through the extraction of a browser fingerprint, to link the fingerprints over time and enable device tracking, as well as the ability to verify the fingerprint's correctness or consistency. We have also presented the countermeasures to browser fingerprinting.

Despite being discovered and popularized relatively recently, browser fingerprinting has attracted the interest of many parties due to its risks to privacy, its potential to replace cookies or other stateful tracking techniques, and its potential uses for security. It has also shown to be resilient to many countermeasures. The technology of browser fingerprinting and the techniques associated are constantly evolving and affect all Web users. This has led to a competitive environment which is simultaneously studied by academics, implemented by professionals, and widely used on many popular websites.

Chapter 3

Uniqueness, Linkability and Consistency

Contents

3.1 Motivations . . . 17
3.2 Overview . . . 18
3.3 Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints . . . 19
3.3.1 Dataset . . . 20
3.3.2 Canvas Fingerprinting . . . 21
3.3.3 Mobile Device Fingerprinting . . . 22
3.3.4 Discussing *Beauty and the Beast* . . . 24
3.4 FP-Stalker : Tracking Browser Fingerprint Evolutions . . . 25
3.4.1 Linking Browser Fingerprints . . . 28
3.4.1.1 Rule-based Linking Algorithm . . . 28
3.4.1.2 Hybrid Linking Algorithm . . . 29
3.4.2 Empirical Evaluation of Fp-Stalker . . . 33
3.4.3 Discussing Fp-Stalker . . . 38
3.5 FP-Scanner: Detecting Browser Fingerprint Inconsistencies . . . 38
3.5.1 Operating System Inconsistencies . . . 39
3.5.2 Browser Inconsistencies . . . 40
3.5.3 Device Inconsistencies . . . 41
3.5.4 Canvas Inconsistencies . . . 41
3.5.5 Empirical Evaluation of FP-Scanner . . . 42
3.5.6 Discussing FP-Scanner . . . 45
3.6 Summary . . . 47

3.1 Motivations

We have studied the properties of *browser fingerprinting* that make it both a risk to privacy and useful for certain applications. The multitude of browsers, devices, configurations and preferences we use to access the Web makes the ecosystem diverse, dynamic and resilient, there's a million moving parts interacting in harmony. But this richness also makes it possible for us, as individuals, through our devices, to be identifiable, discriminable, trackable and verifiable. We have studied browser fingerprinting in detail. First we were drawn by the potential privacy risks of browser fingerprint tracking, and later we were interested in the potential uses, in particular for security. The shear magnitude of the problem limits the potential of doing in-vitro studies of browser fingerprinting, we can't realistically study the millions of devices on the Web in a perfect lab setting. Thus, we have embarked on the use of large-scale empirical studies to shed light on key properties we have identified that make fingerprinting interesting. These properties are fingerprint *uniqueness*, that is, the capacity to uniquely identify a device, fingerprint *linkability*, that is, the capacity to re-identify a device by linking in the future, and fingerprint *consistency*, that is, the

capacity to verify the attributes of a fingerprint. Finally, I would like to mention that the Web, and the browsers we use to access the Web, are constantly changing, advancing, and evolving. This means that, in regards to browser fingerprinting, there's little we can say *absolutely*, but lots we can say at any given time.

3.2 Overview

Uniqueness is a critical property of browser fingerprints. It allows uniquely identifying browsers, devices, and, albeit indirectly, users. It is a necessary part for re-identification and is the property that is most frequently viewed as the danger of fingerprinting. Our paper *Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints* published at S&P 2016 [Laperdrix 2016] focuses on the property of uniqueness and makes the following contributions:

- A novel fingerprinting script that exploits modern web technologies to increase uniqueness. Given the nature of browser fingerprinting and the constant hunt
- We perform the first large-scale study of *canvas fingerprinting*, a now ubiquitous attribute for browser fingerprinting. We show that canvas fingerprinting is one of the most discriminating attributes because it targets the hardware level, including the graphics drivers and card.
- We provide the first large-scale study of mobile device fingerprinting and find that 81% of unique mobile fingerprints in our dataset despite the lack of attributes from desktops, such as plugins and fonts. We show that the wealth of mobile models (different vendors with different firmware versions) result in very rich user-agents and very revealing canvas usage.
- We explore scenarios of possible technological evolutions to improve privacy, and we simulate their impact on browser fingerprinting using our dataset. We find that some changes can drastically reduce uniqueness.

Linkability is the second critical property needed to re-identify browser fingerprints over time. Browser fingerprint tracking requires not only the ability to uniquely identify a browser at a given moment, but the ability identify or link fingerprints from the same device over a period of time. Linking is not assured by uniqueness because some attributes are naturally unstable while others change with system updates, browser updates, driver updates, or a number of configuration changes. Our paper, titled FP-Stalker: *Tracking Browser Fingerprint Evolutions* and published at S&P 2018 [Vastel 2018b], studies the linkability of browser fingerprints and makes the following contributions:

- We highlight the limits of browser fingerprint uniqueness for tracking by showing that fingerprints change frequently (50 % of browser instances changed their fingerprints in less than 5 days, 80 % in less than 10 days).
- We propose two variant algorithms to link fingerprints.
- We compare the accuracy and execution times of our algorithms, and we study how fingerprinting frequency impacts tracking duration.
- We find that around 20% of the devices in our dataset are difficult to track, they're too common and their fingerprints become confused with other devices, while about 26% of the devices can be tracked over the entire duration of their participation in the dataset.

Consistency is the third fundamental property of browser fingerprinting that we have studied. It focuses on how trustworthy, adherent, spoofable or reliable browser fingerprints are. In our particular case, we are interested in the numerous redundant mechanisms to verify and correlate attributes, which allows us to check if a fingerprint is consistent or if it has been altered. Fingerprints rely on multiple layers of software and hardware, and different attributes target different layers, making it particularly difficult to coherently spoof fingerprinting attributes, especially since some attributes that rely on hardware are particularly difficult to coherently alter. In our

paper, *FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies*, published at Usenix Security 2018 [Vastel 2018a], we make the following contributions:

- We propose an approach that leverages the notion of consistency to detect if a fingerprint has been altered.
- We implement a fingerprinting script and an inconsistency scanner capable of detecting altered fingerprints at runtime.
- We test state-of-the-art fingerprinting countermeasures and show how they can be detected using our inconsistency scanner.
- We discuss the impact of our findings, and in particular, we provide insight on when countermeasures become counterproductive to privacy.

The remainder of this section is based on summarized versions of our papers *Beauty and the beast*, FP-Stalker and FP-Scanner, which address the properties of *uniqueness*, *linkability* and *consistency*, respectively.

3.3 Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints

Our paper, *Beauty and the Beast*, could be summed up as a snapshot of the potential risks of fingerprint *uniqueness* in 2015 and 2016. Our study, which is most comparable to the paper published by Eckersley [Eckersley 2010] in PETS 2010, provided information on the *uniqueness* of browser fingerprints, with a detailed analysis of the attributes at hand and how they might be used to identify browsers on the internet. However, such a simple explanation misses a lot of the context and the reasoning put behind this work. At the time, the Web was beginning to see an immense technological push towards new technologies that are now directly integrated into the browser and common place, such as HTML5, the Canvas API, 2D and 3D hardware acceleration, the WebGL API, and countless others. An move towards browsing on mobile devices was taking place, we were talking about them overcoming their desktop counterparts, a quite significant change in personal computing unto itself. We also saw the fall of giants, the Java browser plug-in fell into disarray and was happily banished, and Flash, once the center to pretty much all interactive and dynamic web pages on the Web, was being deprecated and pushed out by the likes of Google and Apple (bye bye Flash cartoons and games). This was the time when the browser was not only the users' window to the Web, but was becoming the heart of what many users consider their personal computing devices. Just six years later we can see how these APIs and new technologies have changed what is possible on the Web, leading to many native applications being displaced and replaced by Web-based variants, once an unimaginable process. This has not stopped either, today we're looking towards new technologies, such as Web assembly, that would provide closer to native performance with better security measures.

The name of our paper comes from the *beauty* of the Web and the new technologies being created to foster it, but also on the hidden dangers of rapidly increasing the attack surface of browsers and opening them up to being exploited, what we are calling the hidden *beast*. To perform our study, we created the AmIUnique project[1] to collect and study browser fingerprints. Our study at the time, which I'll describe throughout this subsection, provided the first large-scale empirical study on the *uniqueness* of browser fingerprints while the Web was moving to new technologies. It showed the dangers of the Canvas API, the effects of deprecating Flash and other NPAPI plugins, and the first data on mobile device fingerprinting, which to our surprise, showed high levels of uniqueness. It also showed the risks of fonts and special characters, emojis, that can be used to recover or verify fingerprint attributes. We simulated scenarios in this paper to show how small or medium changes could reduce uniqueness, it was our attempt to influence towards privacy-friendly technological choices.

[1] https://amiunique.org

3.3.1 Dataset

We launched the AmIUnique website in November 2014 and collected 118,934 browser fingerprints for this study. We implemented a browser fingerprinting script that used the state-of-the-art techniques at the time [Acar 2014, Mowery 2012] and were the first to collect them in a large-scale study. We collected 17 attributes to form a browser fingerprint, of which 89.4% were unique. The complete list of attributes is given in the 'Attribute' column of Table 3.1. The 'Source' column indicates the origin of each attribute (HTTP, JavaScript or Flash). The 'Distinct values' and 'Unique values' columns give a global overview of the most discriminating attributes in a fingerprint. The site is still operational and the fingerprinting script has been frequently updated as new attributes are discovered.

Table 3.1: Browser measurements of AmIUnique fingerprints.

Attribute	Source	Distinct values	Unique values	Value Examples
`User-agent`	HTTP header	11,237	6,559	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36
`Accept`	HTTP header	131	62	text/html,application/xhtml+xml,application /xml;q=0.9,image/webp,*/*;q=0.8
`Encoding`	HTTP header	42	11	gzip, deflate, sdch, br
`Languages`	HTTP header	4,694	2,887	en-US,en;q=0.8,es;q=0.6
`Plugins`	JavaScript	47,057	39,797	Plugin 0: Chrome PDF Viewer; ; mhiehjai. Plugin 1: Chrome PDF Viewer; Portable Document Format; internal-pdf-viewer. Plugin 2: Native Client; ; internal-nacl-plugin.
`Cookies`	JavaScript	2	0	yes
`Local storage`	JavaScript	2	0	no
`Timezone`	JavaScript	55	6	-180
`Resolution`	JavaScript	2,689	1,666	2560x1440x24
`Fonts`	Flash	36,202	31,007	Abyssinica SIL,Aharoni CLM,AR PL UMing CN,AR PL UMing HK, ...
`Headers`	HTTP header	1,182	525	Connection Accept X-Real-IP DNT Cookie Accept-Language Accept-Encoding User-Agent Host
`Platform`	JavaScript	187	99	Linux x86_64
`Do not track`	JavaScript	7	0	yes
`Canvas`	JavaScript	8,375	5,533	
`WebGL Vendor`	Javascript	26	2	NVIDIA Corporation
`WebGL Renderer`	Javascript	1,732	649	NVIDIA GeForce GTX 750 Series
`Ad blocker`	JavaScript	2	0	no

The 'Distinct values' column in Table 3.1 provides the number of different values that we observed for each attribute, while the 'Unique values' column provides the number of values that occurred a single time in our dataset. For example, attributes like the use of cookies or session storage have no unique values since they are limited to "yes" and "no". Other attributes can virtually take an infinite number of values. For example, we observed 6,559 unique values for the user-agent attribute. This is due to the many possible combinations between the browser, its version and the operating system of the device. It is extremely likely that visitors who use an exotic OS with a custom browser, such as Pale Moon on Arch Linux, will present a very rare user-agent, making it possible to be re-identified with just the user-agent. These numbers show that some attributes are more discriminating than others, but they all contribute to building a unique fingerprint.

It's interesting that at the time we observed 2,458 distinct browser plugins, mostly from the now deprecated Netscape Plugin API, assembled in 47,057 different lists of plugins. Many of them were unique and very specialized, sometimes revealing where a person worked or what bank they used. We also observed 221,804 different fonts that we collected through Flash, assembled in 36,202 different lists of fonts. The methods we used to collect these are now deprecated but this shows an incredible amount of diversity to create unique fingerprints.

We observed 222 different HTTP headers, assembled in 1,182 different lists of headers. New headers are added to the standardized ones for many different reasons and from different sources. Some examples include the following browsers adding a special header, or network proxies doing so to tag requests. 182 out of 222 headers appeared in less than 0,1% of the collected fingerprints, and 92 of them come from only one or two fingerprints. These statistics mean that some HTTP headers are highly discriminating and their presence greatly affects the uniqueness of one's fingerprint. Developer's should pay attention to avoid any discriminating information they add to Web requests.

Table 3.2: Summary of statistics

Attr.	Total	<1% FP	<0,1% FP	< 3 FP
Plugin	2,458	2,383 (97%)	2,195 (89%)	950 (39%)
Font	223,498	221,804 (99%)	217,568 (97%)	135,468 (61%)
Header	222	205 (92%)	182 (82%)	92 (41%)

3.3.2 Canvas Fingerprinting

The canvas element in HTML5 [Canvas 2021] allows for the scriptable rendering of 2D shapes and texts. Any website can draw and animate scenes to offer visitors dynamic and interactive content. As discovered by Mowery and al. [Mowery 2012] and investigated by Acar and al. [Acar 2014], canvas fingerprinting can be used to differentiate devices with pixel precision by rendering a specific picture following a fixed set of instructions. This technique gained popularity in tracking scripts because the rendered picture depends on several layers of the system (at least the browser, OS, graphics drivers and hardware). Today, canvas fingerprinting is a staple of browser fingerprinting.

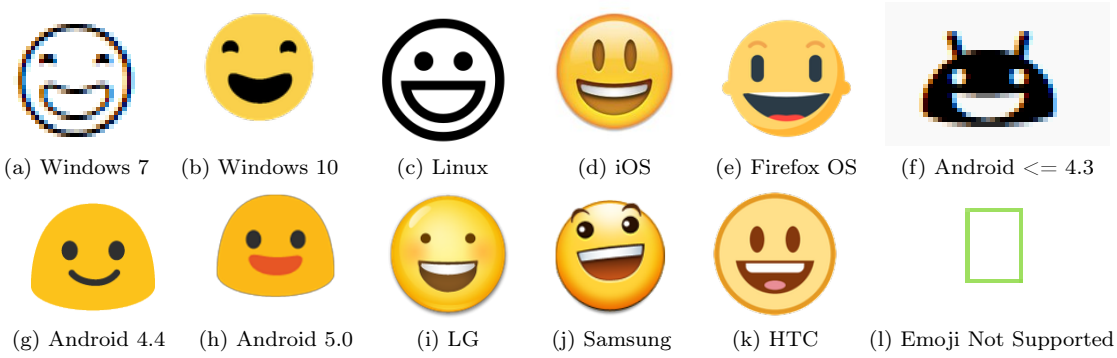
The fingerprinting script used by AmIUnique includes a canvas element. We use it to collect information about three different attributes of the host device. Figure 3.1 displays the image we use, as it is rendered by a Firefox browser running on Fedora 21 with an Intel i7-4600U processor. Our test replicates the test described in detail by Acar et al [Acar 2014]: print a *pangram* twice with different fonts and colors, add the U+1F603 Unicode character and a rectangle with a specific color. We changed the position of the second string so that it is not intertwined with the first. This image captures information from multiple levels.

Figure 3.1: Example of a canvas fingerprint

Font probing. The script tells the browser to render the first *pangram* with a specific font, *Arial*, and the second with a fallback font (we ask for a fake font to trigger the fallback). This makes the image depend on the OS and fonts installed on the device.

Emoji fingerprinting. The last character of our string may be the most important. This is an *emoji* [Unicode 2021]. Officially introduced in the Unicode standard 6.0 in 2010, emojis are ideograms represented by a single Unicode character and are part of the font. Figure 3.2 shows representations of the *Smiling face with open mouth* emoji on different operating systems and

devices. A square means that the browser has not found a single font on the device that supports that emoji. The use of emojis can be a powerful technique, especially on mobile devices where phone manufacturers provide their own sets of emojis.



(a) Windows 7 (b) Windows 10 (c) Linux (d) iOS (e) Firefox OS (f) Android <= 4.3

(g) Android 4.4 (h) Android 5.0 (i) LG (j) Samsung (k) HTC (l) Emoji Not Supported

Figure 3.2: Comparison of the *Smiling face with open mouth* emoji on different devices.

The canvas API rendering depends on the device. In some cases, there are small, pixel-level differences, even when using the same font and emoji. The process to render an image is complex and depends on both hardware and software (e.g. GPU, rendering engine, graphic drivers, anti-aliasing, OS), and this test is affected by variations in any of these layers. In other cases, the differences are quite drastic and obvious. More interestingly, the test is also relatively stable. Figure 3.3 shows a pixel level *diff* between two canvas fingerprints that share the same OS and the same browser but different hardware. We notice pixel shifts on some of the letters and around the emoji.

Figure 3.3: A *diff* of two canvas fingerprints from devices with different hardware but the same browser and OS (differences in red).

Finally, we collected other attributes such as WebGL, DoNotTrack header, ad blocker detection and platform, which each showed varying levels of uniqueness, but did increase stability. The entirety of these attributes strengthen fingerprints, allow detecting inconsistencies and spoofers, and allow identifying mobile devices.

3.3.3 Mobile Device Fingerprinting

Our analysis of mobile device fingerprinting is based on 13,105 mobile fingerprints. We select these fingerprints from our dataset by analyzing the user-agents. If the user-agent contains a substring that is present in a predefined set ('Mobile', 'Android', 'iPhone' or 'iPad'), the fingerprint is selected as a mobile fingerprint, otherwise, it belongs to the desktop/laptop category. Using the attributes from Table 3.1, we succeeded in uniquely identifying 90% of desktop fingerprints. This number is lower for mobile fingerprints at 81%, yet still quite effective. At first sight, the overall results are close. However, the discriminating attributes for mobile fingerprints are very different from those for desktop fingerprints. If we take a look at Figure 3.4, we can notice an important difference. For desktops, more than 37% of the collected fingerprints have a unique list of plugins, while it is at 1% for mobile devices. Mobile devices do not support browser plugins. For example, Adobe removed the Flash player from the Google Play store in August 2012 as part of a change

of focus for the company [Adobe 2012]. Plugins are considered unsuitable for the modern web and Google states in their move to deprecate NPAPI support from Chrome that these plugins are a source of "hangs, crashes, security incidents, and code complexity" [Schuh 2013]. This helps mobile device users gain some privacy with regards to fingerprint uniqueness. The level of entropy of the plugin attribute is close to zero (some iOS systems do have the QuickTime plugin and some Android systems reported having Flash, possibly from legacy installations). The lack of plugins also reduces information leaks that could come from them. In particular, the attributes leaked through the Flash API are unavailable.

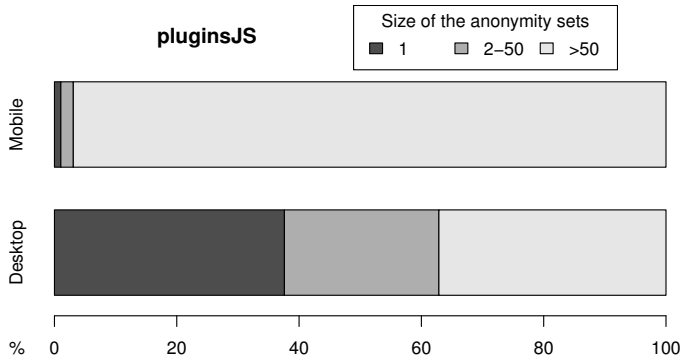


Figure 3.4: Comparison of anonymity set sizes on the list of plugins between desktop and mobile devices

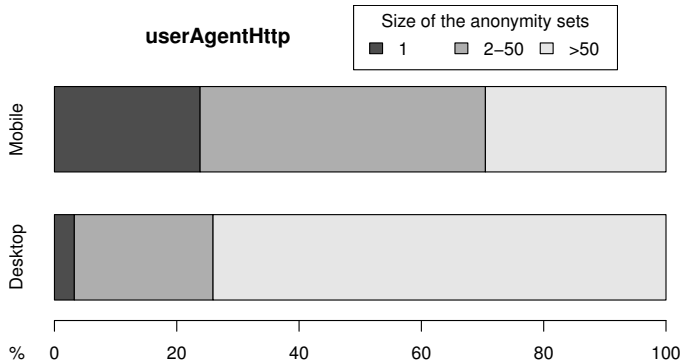


Figure 3.5: Comparison of the user agent anonymity set sizes on desktop and mobile devices

Despite the unavailability fonts lists and plugins, mobile fingerprints are still very much distinguishable. This is due to two main factors: **very rich and revealing user agents and very discriminating emojis**. Figure 3.5 shows that user agents on mobile devices are five times more unique than on desktops. In our dataset, about 1 smartphone out of 4 is instantaneously recognizable with just the user agent. This is due to two factors: phone manufacturers including firmware or device models and versions directly in the user agent, or applications changing the user agent when browsing through the application. In some cases, the carrier sells a special version of the phone, and this is also indicated, revealing the user's carrier (a "SM-G900P" was reported, this is a Samsung Galaxy S5 and the "P" is unique to the Sprint phone carrier). An example of firmware version being added:

```
Mozilla/5.0 (Linux; Android 5.0.1; Nexus 5 Build/LRX22C) AppleWebKit/537.36 (KHTML, like
↪  Gecko) Chrome/40.0.2214.109 Mobile Safari/537.36
```

And an example when browsing through the Facebook app, where the phone carrier (Vodafone UK) and the exact model of the phone is included in the user agent:

```
Mozilla/5.0 (iPhone; CPU iPhone OS 8_1_1 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like
↪ Gecko) Mobile/12B436[FBAN/FBIOS;FBAV/20.1.0.15.10;FBBV/5758778;FBDV/iPhone7,2;
↪ FBMD/iPhone;FBSN/iPhone OS;FBSV/8.1.1;FBSS/2;FBCR/vodafoneUK;FBLC/en_GB;FBOP/5]
```

The second highest source of entropy for mobile devices comes from canvas fingerprinting. Mobiles have unique hardware, fonts and emojis. These can impact the rendered picture as explained in section 3.3.2. As seen in Figure 3.2, manufacturers customize emojis, even between different versions or models of devices, splitting their user base into recognizable groups. We saw that user-agents can give very discriminating information on the user's device. Some smarpthones running Android give the exact model and firmware version of their phone. Looking at Figure 3.6, user agents from the Chrome mobile browser are ten times more unique than user agents from the Firefox browser (40% against less than 4%). This can in part be explained by the fact that Chrome it is automatically installed on Android devices. When a phone manufacturer builds its tailored firmware to be delivered to its clients, the embedded Chrome browser is given a user agent with the corresponding phone model and Android version. Firefox, which can be downloaded and installed, does not contain this type of information because the store only offers a generic version for every device and it does not change its user agent during installation. Firefox indirectly provides better protection against fingerprint tracking by not disclosing device-related information. Here are wo fingerprints collected from the same device but with Chrome then Firefox:

```
//Chrome
Mozilla/5.0 (Linux; Android 4.4.4; D5803 Build/23.0.1.A.5.77) AppleWebKit/537.36 (KHTML, like
↪ Gecko) Chrome/39.0.2171.93 Mobile Safari/537.36
//Firefox
Mozilla/5.0 (Android; Mobile; rv:34.0) Gecko/34.0 Firefox/34.0
```

Figure 3.7 shows the size of anonymity sets for user agents on both Android and iOS devices. We can see that user agents on Android are more diverse, with three times as many users being in an anonymity set of size 1 (9% for iOS devices and 35% for Android devices). This is due to the wealth of Android models available on the market. Moreover, our dataset may not be representative of the diversity of Android devices so these percentages may be higher. For iOS devices, the diversity is still high but much less pronounced since users share more devices with identical configurations.

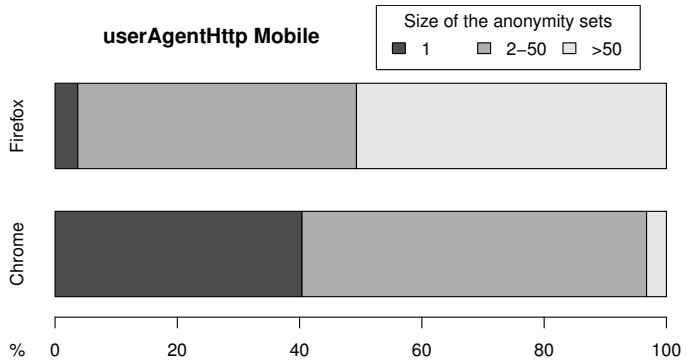


Figure 3.6: Comparison of anonymity set sizes of user agents from Chrome and Firefox on mobile devices

3.3.4 Discussing *Beauty and the Beast*

We analyzed 118,934 browser fingerprints collected through the AmIUnique.org web site to understand the impact recent technological changes were having on browser fingerprint uniqueness. We argue that modern web technologies provide a much improved user experience, albeit to the detriment of privacy. We found that despite improvements to user experience and the removal of

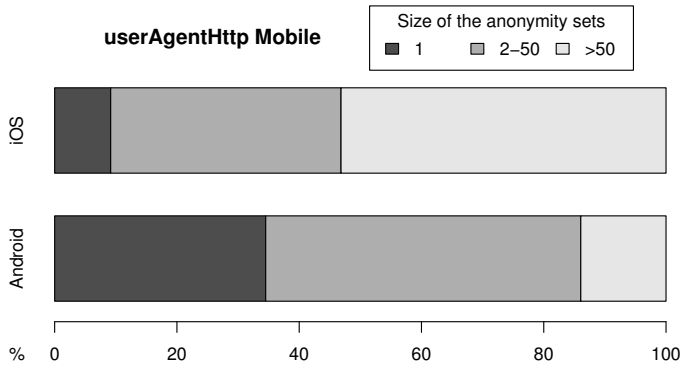


Figure 3.7: Comparison of anonymity set sizes of user agents from Android and iOS devices

highly detrimental technologies to privacy, such as the once ubiquitous Flash plugin, new APIs had increased the browser's attack surface and the uniqueness of browser fingerprints remains high. We performed the first large-scale study of canvas fingerprinting and showed the technique to be quite useful at identifying devices.

We also provided the first extensive analysis of fingerprints collected from mobile devices: 81% of the mobile fingerprints in our dataset are unique. We show that HTTP headers and HTML5 canvas fingerprinting play an essential role in identifying browsers on these devices. This was a surprise since mobile browsers had little support of plugins, which were the main risk to privacy, and they are generally less customizable. We had expected the sheer number of identical devices being sold to limit fingerprint uniqueness. Spooren et al. analyzed 59 mobile device fingerprints [Spooren 2015] in 2015, one year before our work, and concluded that *"the fingerprints taken from mobile devices are far from unique"*. Furthermore, in the absence of the Flash plugin to provide the list of fonts, and the absence of NPAPI plugins, there is no longer any major discriminating attributes, thus identification is based on the collection of many lesser attributes that appear harmless by themselves, but when aggregated lead to unique fingerprints. In the end, desktop and mobile fingerprints are similarly unique even though the discriminating information does not originate from the same attributes.

3.4 FP-Stalker : Tracking Browser Fingerprint Evolutions

FP-Stalker focuses on the linkability of browser fingerprints over time. Long-term tracking requires not only unique browser fingerprints, but must also link fingerprints that originate from the same browser instance over time. The rest of this section is a summarized version of our paper FP-Stalker: *Tracking Browser Fingerprint Evolutions* published at S&P 2018 [Vastel 2018b].

Dataset. Through the AmIUnique project, we collected a raw dataset of 172,285 fingerprints from 7,965 different installations (we call these *browser instances*). All browser fingerprints were obtained from the AmIUnique extensions for Chrome and Firefox installed from July 2015 to August 2017 by participants in this study. The extensions load a page in the background that fingerprints the browser every 4 hours. Compared to a fingerprinting website, the only additional information we collect is a unique identifier we generate per installation that serves to establish the ground truth. After processing these rules, we obtain a final dataset of 98,598 fingerprints from 1,905 browser instances. Figure 3.8 presents the number of fingerprints and distinct browser instances per month over the two year period.

Figure 3.9 illustrates the anonymity set sizes against the number of participants involved in this study. The long tail reflects that 99 % of the browser fingerprints are unique among all the participants and belong to a single browser instance, while only 10 browser fingerprints are shared by more than 5 browser instances.

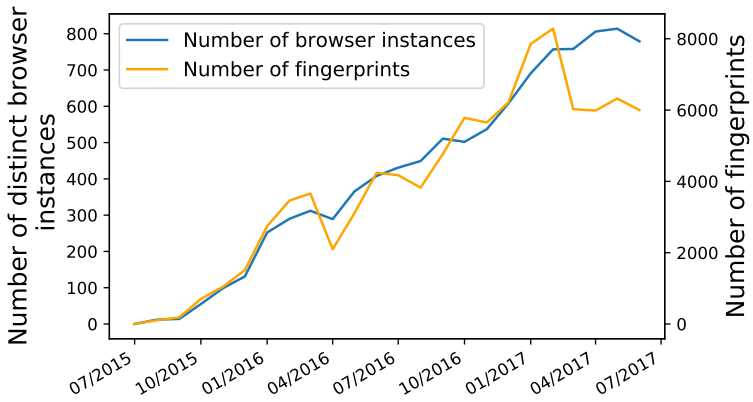


Figure 3.8: Number of fingerprints and distinct browser instances per month

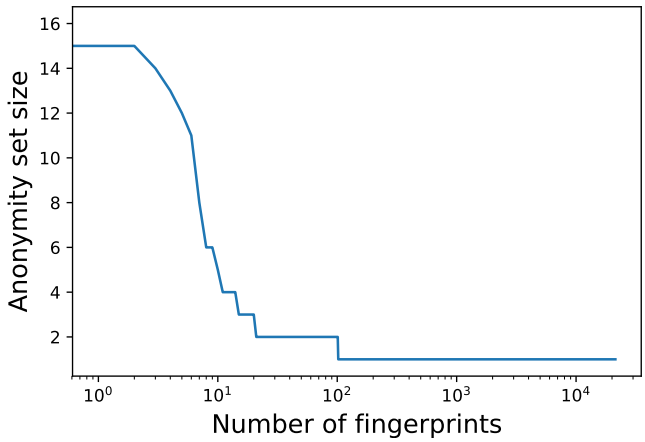


Figure 3.9: Browser fingerprint anonymity set sizes

Evolution triggers. Browser fingerprints naturally evolve for several reasons. We identified the following categories of changes:

- Automatic evolutions happen without direct user intervention. This is mostly due to software upgrades.
- Context-dependent evolutions being caused by changes in the user's context. Some attributes, such as `resolution` or `timezone`, are impacted by a contextual change, such as connecting a computer to an external screen or traveling to a different timezone.
- User-triggered evolutions that require an action from the user. They concern configuration-specific attributes, such as `cookies`, `do not track` or `local storage`.

To know how long attributes remain constant and if their stability depends on the browser instance, we compute the average time, per browser instance, that each attribute does not change. Table 3.3 presents the median, the 90^{th} and 95^{th} percentiles of the duration each attribute remains constant, on average, in browser instances. We observe that the `User agent` is rather unstable in most browser instances as its value is systematically impacted by software updates. In comparison, attributes such as `cookies`, `local storage` and `do not track` rarely change if ever. Moreover, we observe that attributes evolve at different rates depending on the browser instance. For example, `canvas` remains stable for 290 days in 50% of the browser instances, whereas it changes every 17.2 days for 10% of them. The same phenomena can be observed for the `screen resolution` where more than 50% of the browser instances never see a change, while 10% change every 3.1 days on average. More generally this points to some browser instances being quite stable, and thus, more trackable, while others aren't.

Table 3.3: Durations the attributes remained constant for the median, the 90^{th} and the 95^{th} percentiles.

		Percentile (days)		
Attribute	**Trigger**	**50th**	**90th**	**95th**
`Resolution`	Context	Never	3.1	1.8
`User agent`	Automatic	39.7	13.0	8.4
`Plugins`	Automatic/User	44.1	12.2	8.7
`Fonts`	Automatic	Never	11.8	5.4
`Headers`	Automatic	308.0	34.1	14.9
`Canvas`	Automatic	290.0	35.3	17.2
`Major browser version`	Automatic	52.2	33.3	23.5
`Timezone`	Context	206.3	53.8	26.8
`Renderer`	Automatic	Never	81.2	30.3
`Vendor`	Automatic	Never	107.9	48.6
`Language`	User	Never	215.1	56.7
`Dnt`	User	Never	171.4	57.0
`Encoding`	Automatic	Never	106.1	60.5
`Accept`	Automatic	Never	163.8	109.5
`Local storage`	User	Never	Never	320.2
`Platform`	Automatic	Never	Never	Never
`Cookies`	User	Never	Never	Never

Evolution frequency. Another key indicator to observe is the elapsed time (E_t) before a change occurs in a browser fingerprint. Figure 3.10 depicts the cumulative distribution function of E_t for all fingerprints (blue), or averaged per browser instance (orange). After one day, at least one transition occurs in 45.2 % of the observed fingerprints. The 90^{th} percentile is observed after 13 days and the 95^{th} percentile after 17.3 days. This means the probability that at least one transition occurs in 13 days is 0.9 (blue). It is important to point out that changes occur more or less frequently depending on the browser instance (orange). While some browser instances change often (20% change in less than two days) others are much more stable (23% have no changes after 10 days). Keeping pace with the frequency of change is a challenge for browser fingerprint linking algorithms.

Evolution rules. While it is difficult to anticipate browser fingerprint evolutions, we can observe how individual attributes evolve. In particular, evolutions of the `User agent` are often tied to browser upgrades. Nevertheless, not all attribute changes can be explained in this manner, some values are difficult to anticipate. For example, the value of the `canvas` attribute is the result of an image rendered by the browser instance and depends on many different software and hardware layers. The same applies, although to a lesser extent, to screen `resolution`, which can take unexpected values depending on the connected screen. Based on these observations, the accuracy of linking browser fingerprint evolutions depends on the inference of such evolution rules. We first identified rules empirically, and then learned others automatically to achieve an efficient algorithm to track browser fingerprints over time.

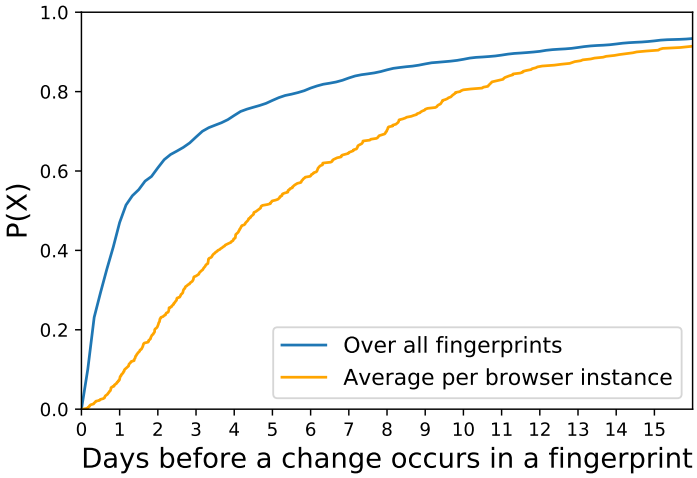


Figure 3.10: CDF of the elapsed time before a fingerprint evolution for all the fingerprints, and averaged per browser instance.

3.4.1 Linking Browser Fingerprints

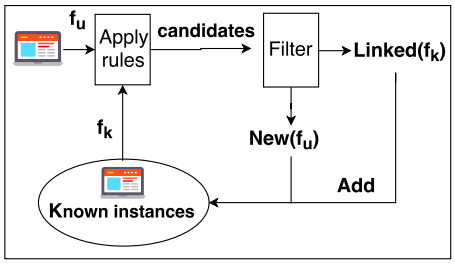
In Fp-Stalker, we have implemented two variant algorithms to link browser fingerprints, as depicted in Figure 3.11. The first is a rule-based algorithm that uses a static ruleset, and the second is a hybrid algorithm that combines both rules and machine learning. Our results show that the rule-based algorithm is faster but the hybrid algorithm is more precise while still maintaining acceptable execution times.

When collecting browser fingerprints, it is possible that a fingerprint comes from a previous visitor—*i.e.*, a known browser instance—or from a new visitor—*i.e.*, an unknown browser instance. The objective of fingerprint linking is to match fingerprints to their browser instance and follow the browser instance as long as possible by linking all of its fingerprint evolutions. In the case of a match, linked browser fingerprints are given the same identifier, which means the linking algorithm considers that they originate from the same browser instance. If the browser fingerprint cannot be linked, the algorithm assigns a new identifier to the fingerprint.

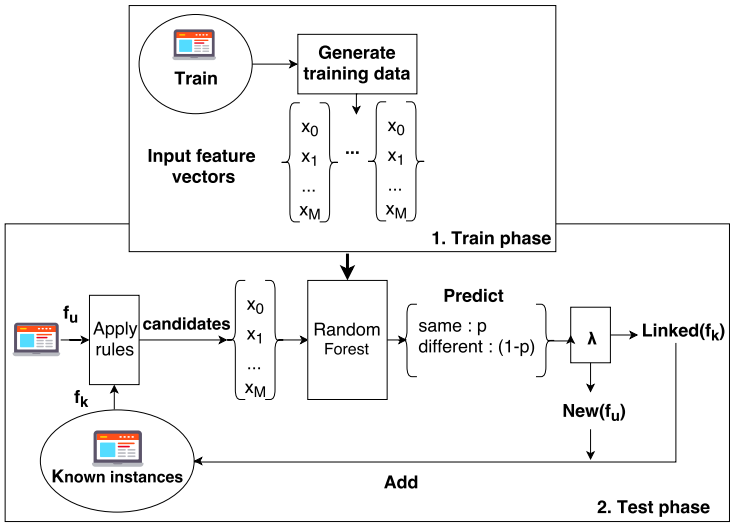
More formally, given a set of known browser fingerprints F, each $f \in F$ has an identifier $f.id$ that links to the browser instance it belongs to. Given an unknown fingerprint $f_u \notin F$ for whom we ignore the real id, a linking algorithm returns the browser instance identifier $f_k.id$ of the fingerprint f_k that maximizes the probability that f_k and f_u belong to the same browser instance. This computation can be done either by applying rules, or by training an algorithm to predict this probability. If no known fingerprint can be found, it assigns a new id to f_u. For optimization purposes, we only hold and compare the last ν fingerprints of each browser instance b_i in F. The reason is because if we linked, for example, 3 browser fingerprints f_A, f_B and f_C to a browser instance b_i then, when trying to link an unknown fingerprint f_u, it is rarely useful to compare f_u to the oldest browser fingerprints of b_i. That is, newer fingerprints are more likely to produce a match, hence we avoid comparing old fingerprints in order to improve execution times. In our case we set the value of ν to 2.

3.4.1.1 Rule-based Linking Algorithm

The first variant of Fp-Stalker is a rule-based algorithm that uses static rules designed from attribute stability presented in Table 3.3 to determine if an unknown fingerprint f_u belongs to the same browser instance as a known fingerprint f_k. We also define rules based on constraints that we would not expect to be violated, such as, a browser's family should be constant (e.g., the same browser instance cannot be Firefox one moment and Chrome at a later time), the Operating System is constant, and the browser version is either constant or increases over time. The full



(a) Rule-based variant of Fp-Stalker. Uses a set of static rules to determine if fingerprints should be linked to the same browser instance or not.



(b) Hybrid variant of Fp-Stalker. The training phase is used to learn the probability that two fingerprints belong to the same browser instance, and the testing phase uses the random forest-based algorithm to link fingerprints.

Figure 3.11: Fp-Stalker: Overview of both algorithm variants. The rule-based algorithm is simpler and faster but the hybrid algorithm leads to better fingerprint linking.

list of rules are presented in the paper.

In order to link f_u to a fingerprint f_k, we apply the rules to each known fingerprint taken from F. As soon as a rule is not matched, the known fingerprint is discarded and we move onto the next. If a fingerprint matches all the rules it is added to a list of potential candidates, *candidates*. Moreover, in case fingerprints f_k and f_u are identical, we add it to the list of exact matching candidates, *exact*. Once the rule verification process is completed, we look at the two lists of candidates. If *exact* is not empty, we check if there is only one candidate or if all the candidates come from the same browser instance. If it is the case, then we link f_u with this browser instance, otherwise we assign a new id to f_u. In case no exact candidate is found, we look at *candidates* and apply the same technique as for *exact*. We summarize the rule-based approach in Algorithm 1.

3.4.1.2 Hybrid Linking Algorithm

The second variant of Fp-Stalker mixes the rule-based algorithm with machine learning to produce a hybrid algorithm. It uses a set of rules from the previous algorithm regarding the OS, the browser's platform and version, since we consider them constraints that should not be violated between two fingerprints of a same browser instance. However, for other rules, the situation is

Algorithm 1 Rule-based matching algorithm

```
function FingerprintMatching(F, f_u)
    rules = {rule_1, ..., rule_6}
    candidates ← ∅
    exact ← ∅
    for f_k ∈ F do
        if VerifyRules(f_k, f_u, rules) then
            if nbDiff = 0 then
                exact ← exact ∪ ⟨f_k⟩
            else
                candidates ← candidates ∪ ⟨f_k⟩
            end if
        end if
    end for
    if |exact| > 0 and SameIds(exact) then
        return exact[0].id
    else if |candidates| > 0 and SameIds(candidates) then
        return candidates[0].id
    else
        return GenerateNewId()
    end if
end function
```

SameIds is a function that, given a list of candidates, returns true if all of them share the same id, else false.

more fuzzy. It is not as clear when to allow attributes to be different, how many of them can be different, and with what dissimilarity. Instead of manually crafting rules for each of these attributes, we propose to use machine learning to discover them. The interest of combining both rules and machine learning approaches is that rules are faster than machine learning, but machine learning tends to be more precise. Thus, by applying the rules first, it helps keep only a subset of fingerprints on which to apply the machine learning algorithm.

The first step of this algorithm is to apply rules on f_u and all $f_k \in F$. We keep the subset of browser fingerprints f_{ksub} that verify these rules. If, during this process, we found any browser fingerprints that exactly match f_u, then we add them to *exact*. In case *exact* is not empty and all of its candidates are from the same browser instance, we stop here and link f_u with the browser instance in *exact*. Otherwise, if there are multiple exact candidates but from different browser instances, then we assign a new browser id to f_u. If the set of exact candidates is empty, we continue with a second step that leverages machine learning. In this step, for each fingerprint $f_k \in f_{ksub}$, we compute the probability that f_k and f_u come from the same browser instance using a random forest model. We keep a set of fingerprint candidates whose probability is greater than a λ threshold parameter. If the set of candidates is empty, we assign a new id to f_u. Otherwise, we keep the set of candidates with the highest and second highest probabilities, c_{h1} and c_{h2}. Then, we check if c_{h1} contains only one candidate or if all of the candidates come from the same browser instance. If it is not the case, we check that either the probability p_{h1} associated with candidates of c_{h1} is greater than the probability p_{h2} associated with candidates of $c_{h2} + diff$, or that c_{h2} and c_{h1} contains only candidates from the same browser instance. Algorithm 2 summarizes the hybrid approach.

Computing the probability that two fingerprints f_u and f_k originate from the same browser instance can be modeled as a binary classification problem where the two classes to predict are `same browser instance` and `different browser instance`. We use the random forest algorithm [Breiman 2001] to solve this binary classification problem. A random forest is an ensemble learning method for classification that operates by constructing a multitude of decision trees at training time and outputting the class of the individual trees. In the case of Fp-Stalker, each decision tree makes a prediction and votes if the two browser fingerprints come from the

Algorithm 2 Hybrid matching algorithm

```
function FingerprintMatching(F, f_u, λ)
    rules = {rule_1, rule_2, rule_3}
    exact ← ∅
    F_ksub ← ∅
    for f_k ∈ F do
        if VerifyRules(f_k, f_u, rules) then
            if nbDiff = 0 then
                exact ← exact ∪ ⟨f_k⟩
            else
                F_ksub ← F_ksub ∪ ⟨f_k⟩
            end if
        end if
    end for
    if |exact| > 0 then
        if SameIds(exact) then
            return exact[0].id
        else
            return GenerateNewId()
        end if
    end if
    candidates ← ∅
    for f_k ∈ F_ksub do
        ⟨x_1, x_2, ..., x_M⟩ = FeatureVector(f_u, f_k)
        p ← P(f_u.id = f_k.id | ⟨x_1, x_2, ..., x_M⟩)
        if p ≥ λ then
            candidates ← candidates ∪ ⟨f_k, p⟩
        end if
    end for
    if |candidates| > 0 then
        c_h1, p_h1 ← getCandidatesRank(candidates, 1)
        c_h2, p_h2 ← getCandidatesRank(candidates, 2)
        if SameIds(c_h1) and p_h1 > p_h2 + diff then
            return candidates[0].id
        end if
        if SameIds(c_h1 ∪ c_h2) then
            return candidates[0].id
        end if
    end if
    return GenerateNewId()
end function
```

getCandidatesRank is a function that given a list of candidates and an rank i, returns a list of candidates with the ith greatest probability, and this probability.

same browser instance. The result of the majority vote is chosen. Our main motivation to adopt a random forest instead of other classifiers is because it provides a good tradeoff between precision and the interpretation of the model. In particular, the notion of feature importance in random forests allows Fp-Stalker to interpret the importance of each attribute in the decision process. In summary, given two fingerprints, $f_u \notin F$ and $f_k \in F$, whose representation is reduced to a single feature vector of M features $X = \langle x_1, x_2, ..., x_M \rangle$, where the feature x_n is the comparison of the attribute n for both fingerprints (the process of transforming two fingerprints into a feature vector is presented after). Our random forest model computes the probability $P(f_u.id = f_k.id \mid (x_1, x_2, ..., x_M))$ that f_u and f_k belong to the same browser instance.

Input Feature Vector. To solve the binary classification problem, we provide an input vector $X = \langle x_1, x_2, ..., x_M \rangle$ of M features to the random forest classifier. The features are mostly

Table 3.4: Feature importances of the random forest model calculated from the train set.

Rank	Feature	Importance
1	`Number of changes`	0.350
2	`Languages HTTP`	0.270
3	`User agent HTTP`	0.180
4	`Canvas`	0.090
5	`Time difference`	0.083
6	`Plugins`	0.010
7	`Fonts`	0.008
8	`Renderer`	0.004
9	`Resolution`	0.003

pairwise comparisons between the values of the attributes of both fingerprints (*e.g.*, `Canvas`, `User agent`). Most of these features are binary values (`0` or `1`) corresponding to the equality or inequality of an attribute, or similarity ratios between these attributes. We also include a `number of changes` feature that corresponds to the total number of different attributes between f_u and f_k, as well as the time difference between the two fingerprints. We obtained a feature vector composed of the attributes presented in Table 3.4 after making a feature selection. Interestingly, we see that the most important feature is the number of differences between two fingerprints, and the second most discriminating attribute is the list of languages. Although this may seem surprising since the list of languages does not have high entropy, it does remain stable over time, as shown in Table 3.3, which means that if two fingerprints have different languages, this often means that they do not belong to the same browser instance.

Training Random Forests. We train the random forest classifier to estimate the probability that two fingerprints belong to the same browser instance. To do so, we split our dataset chronologically into two sets: a training set and a test set. The training set is composed of the first 40 % of fingerprints, and the test set of the last 60 %. The random forest detects fingerprint evolutions by computing the evolutions between fingerprints as feature vectors. During the training phase, it needs to learn about correct evolutions by computing relevant feature vectors from the training set. Algorithm 3 describes this training phase, which is split into two steps.

Algorithm 3 Compute input feature vectors for training

```
function BuildTrainingVectors(ID,F,δ,ν)
    T ← ∅
    for id ∈ ID do                                          ▷ Step 1
        F_id ← BrowserFingerprints(id, F)
        for f_t ∈ F_id do
            T ← T ∪ FeatureVector(f_t, f_{t−1})
        end for
    end for
    for f ∈ F do                                            ▷ Step 2
        f_r ← random(F)
        if f.id ≠ f_r.id then
            T ← T ∪ FeatureVector(f, f_r)
        end if
    end for
    return T
end function
```

In `Step 1`, for every browser instance (id) of the training set, we compare each of its finger-

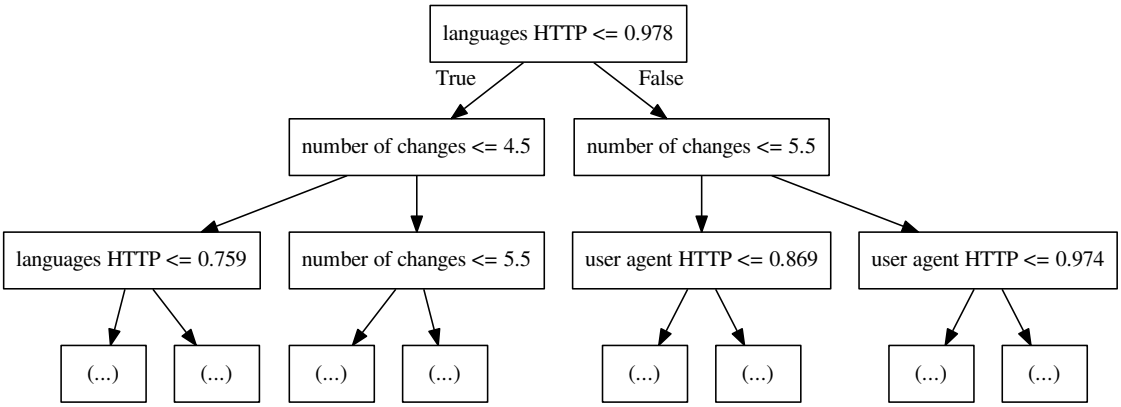


Figure 3.12: First 3 levels of a single tree classifier from our forest.

prints ($f_t \in$ BROWSERFINGERPRINTS(id, F)) present in the training set (F) with the previous one (f_{t-1}). By doing so, FP-STALKER captures the atomic evolutions that occur between two consecutive fingerprints from the same browser instance. We apply BUILDTRAININGVECTORS() for different collect frequencies (time difference between t and $t-1$) to teach our model to link fingerprints even when they are not equally spaced in time. While `Step 1` teaches the random forest to identify fingerprints that belong to the same browser instance, it is also necessary to identify when they do not. `Step 2` compares fingerprints from different browser instances. Since the number of fingerprints from different browser instances is much larger than the number of fingerprints from the same browser instance, we limit the number of comparisons to one for each fingerprint. This technique is called *undersampling* [Loyola-González 2013] and it reduces overfitting by adjusting the ratio of input data labeled as *true*—*i.e.*, 2 fingerprints belong to the same browser instance— against the number of data labeled as *false*—*i.e.*, 2 fingerprints are from different browser instances. Otherwise, the algorithm would tend to simply predict *false*.

After training our random forest classifier, we obtain a forest of decision trees that predict the probability that two fingerprints belong to the same browser instance. Figure 3.12 illustrates the first three levels of one of the decision trees. These levels rely on the `languages`, the `number of changes` and the `user agent` to make a decision. If an attribute has a value below its threshold, the decision path goes to the left child node, otherwise it goes to the right child node. The process is repeated until we reach a leaf of the tree. The prediction corresponds to the class (same/different browser instance) that has the most instances over all the leaf nodes.

3.4.2 Empirical Evaluation of FP-STALKER

We asses FP-STALKER's capacity to *i)* correctly link fingerprints from the same browser instance, and to *ii)* correctly predict when a fingerprint belongs to a browser instance that has never been seen before. Both variants are effective in linking fingerprints, however, the rule-based variant is faster while the hybrid variant is more precise. Figure 3.13 illustrates the linking and evaluation process with a scenario. Our database contains perfect tracking chains because of the unique identifiers our extensions use to identify browser instances. From there, we sample the database using different collection frequencies and generate a test set that removes the identifiers, resulting in a mix of fingerprints from different browsers. The resulting test set is then run through FP-STALKER to reconstruct the best browser instance chains as possible.

A *tracking chain* is a list of fingerprints that have been linked—*i.e.*, fingerprints for which the linking algorithm assigned the same identifier. A chain may be composed of one or more fingerprints. In case of a perfect linking algorithm, each browser instance would have a unique tracking chain—*i.e.*, all of its fingerprints are grouped together and are not mixed with fingerprints from any other browser instances. However, in reality, fingerprinting is a statistical attack and mistakes may occur during the linking process, which means that:

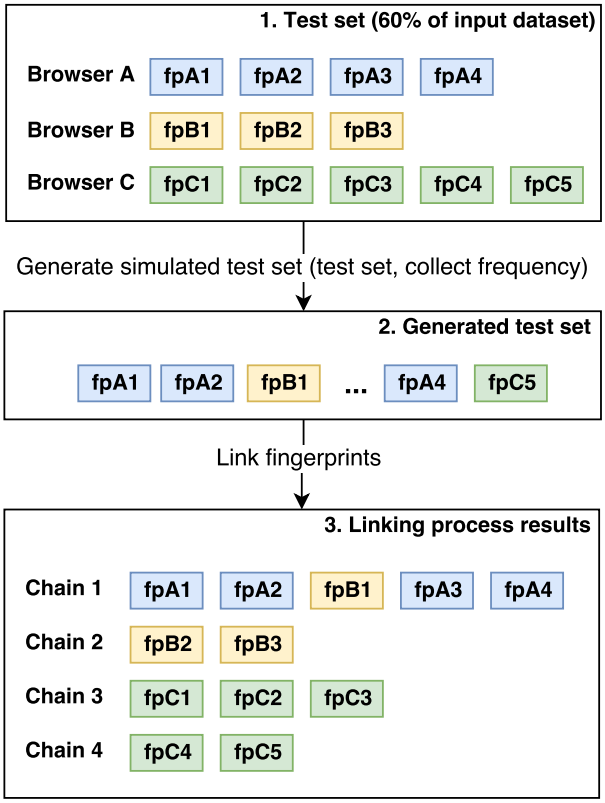


Figure 3.13: Overview of our evaluation process that allows testing the algorithms using different simulated collection frequencies.

1. Fingerprints from different browser instances may be put in the same tracking chain.
2. Fingerprints from a given browser instance may be split into different tracking chains.

The lower part of Figure 3.13 shows examples of these mistakes. *Chain 1* has an incorrect fingerprint `fpB1` from `Browser B`, and `chains 3` and `4` contain fingerprints from `browser C`—*i.e.*, `fpC3` and `fpC4` were not linked leading to a split).

We present the *tracking duration* metric to evaluate the capacity of an algorithm to track browser instances over time. We define *tracking duration* as the period of time a linking algorithm matches the fingerprints of a browser instance within a single tracking chain. More specifically, the tracking duration for a browser $\texttt{b}_i$ in a chain $\texttt{chain}_k$ is defined as $CollectFrequency \times (\#b_i \in chain_k - 1)$. We subtract one because we consider a browser instance to have been tracked, by definition, from the second linked fingerprint onwards.

The *average tracking duration* for a browser instance b_i is the arithmetic mean of its tracking duration across all the tracking chains the instance is present in. For example, in Figure 3.13, in the case of `browser B` the tracking duration of `browser B` in `chain 1` is $0 \times CollectFrequency$, and the tracking duration in `chain 2` is $1 \times CollectFrequency$, thus the average tracking duration is $0.5 \times CollectFrequency$. In the same manner, the *average tracking duration* of *browser C* is $1.5 \times CollectFrequency$.

The *maximum tracking duration* for a browser instance b_i is defined as the maximum tracking duration across all of the tracking chains the browser instance is present in. In the case of `browser C`, the maximum tracking duration occurred in `chain 3` and is equal to $2 \times CollectFrequency$.

The *Number of assigned ids* represents the number of different identifiers that have been assigned to a browser instance by the linking algorithm. It can be seen as the number of tracking chains in which a browser instance is present. For each browser instance, a perfect linking algorithm would group all of the browser's fingerprints into a single chain. Hence, each browser instance would have a *number of assigned ids* of 1. Figure 3.13 shows an imperfect case where

`browser C` has been assigned 2 different ids (`chain 3` and `chain 4`).

The *ownership ratio* reflects the capacity of an algorithm to not link fingerprints from different browser instances. The `owner` of a tracking chain $\texttt{chain}_k$ is defined as the browser instance $\texttt{b}_i$ that has the most fingerprints in the chain. Thus, we define *ownership ratio* as the number of fingerprints that belong to the `owner` of the chain divided by the length of the chain. For example, in `chain 1`, `browser A` owns the chain with an `ownership ratio` of $\frac{4}{5}$ because it has 4 out of 5 fingerprints. In practice, an *ownership ratio* close to 1 means that a tracking profile is not polluted with information from different browser instances.

Simulating the collect frequency. To evaluate the effectiveness of Fp-Stalker we start from our test set of 59,159 fingerprints collected from 1,395 browser instances (60% of our input dataset, see Section 3.4.1.2). However, we do not directly use this set. Instead, by sampling the test set, we generate new, smaller datasets using a configurable collect frequency. Because our dataset is fine-grained, it allows us to simulate the impact fingerprinting frequency has on tracking. The intuition being that if a browser is fingerprinted less often, it becomes harder to track.

To generate a dataset for a given collect frequency, we start from the test set of 59,159 fingerprints, and, for each browser instance, we look at the collection date of its first fingerprint. Then, we iterate in time with a step of $collect\_frequency$ days and recover the browser instance's fingerprint at time $t + collect\_frequency$. It may be the same fingerprint as the previous collect or a new one. We do this until we reach the last fingerprint collected for that browser id. This allows us to record a sequence of fingerprints that correspond to the sequence a fingerprinter would obtain if the browser instance was fingerprinted at a frequency of $collect\_frequency$ days. The interest of sampling is that it is more realistic than using all of the fingerprints from our database since they are very fine-grained. Indeed, the extension is capable of catching even short-lived changes (*e.g.*, connecting an external monitor), which is not always possible in the wild. Finally, it allows us to investigate how fingerprint collection frequency impacts browser tracking. Figure 3.14 provides an example of the process to generate a dataset with a $collect\_frequency$ of two days. Table 3.5 presents, for each simulated collect frequency, the number of fingerprints in the generated test sets.

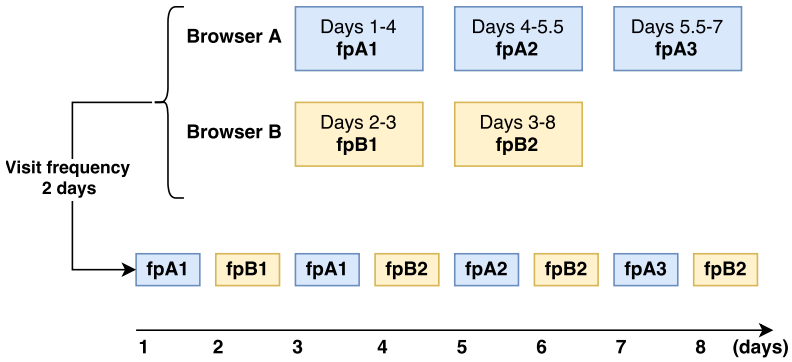


Figure 3.14: Example of the process to generate a simulated test set. The dataset contains fingerprints collected from browser's `A` and `B`, which we sample at a $collect\_frequency$ of 2 days to obtain a dataset that allows us to test the impact of $collect\_frequency$ on fingerprint tracking.

Tracking duration. Figure 3.15 plots the average *tracking duration* against the collect frequency for the three algorithms. On average, browser instances from the test set were present for 109 days, which corresponds to the maximum value our linking algorithm could potentially achieve. We see that the hybrid variant of Fp-Stalker is able to, on average, keep track of browser instances for a longer period of time than the rule-based variant. We also compare to the Panopticlick algorithm [Eckersley 2010]. In the case where a browser gets fingerprinted every

Table 3.5: Number of fingerprints per generated test set after simulating different collect frequencies

Collect frequency (days)	Number of fingerprints
1	171,735
2	86,225
3	57,695
4	43,441
5	34,916
6	29,195
7	25,155
8	22,065
10	17,814
15	12,100
20	9,259

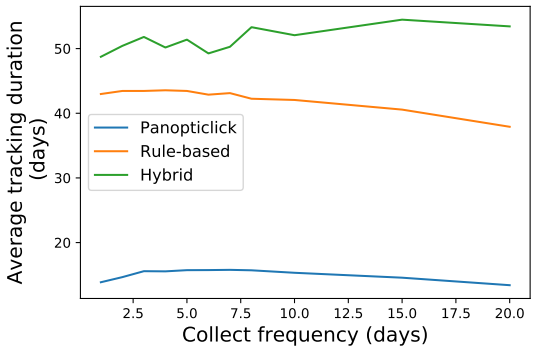


Figure 3.15: Average *tracking duration* against simulated collect frequency for the three algorithms

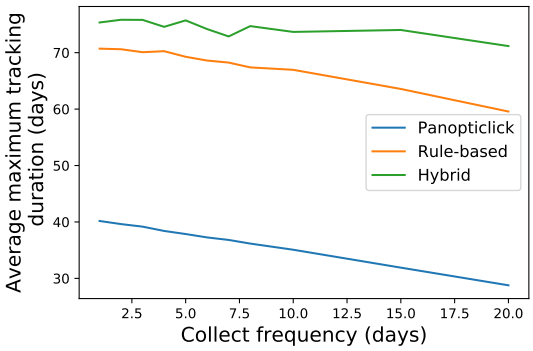


Figure 3.16: Average maximum tracking duration against simulated collect frequency for the three algorithms. This shows averages of the longest tracking durations that were constructed.

three days, Fp-Stalker can track it for 51,8 days, on average. More generally, the hybrid variant of Fp-Stalker has an average tracking duration of about 9 days more than the rule-based variant and 15 days more than the Panopticlick algorithm.

Figure 3.16 presents the average *maximum tracking duration* against the collect frequency for the three algorithms. We see that the hybrid algorithm outperforms the two other algorithms because the it constructs longer tracking chains with less mistakes. On average, the maximum average tracking duration for Fp-Stalker's hybrid version is in the order of 74 days, meaning that at most users were generally tracked for this duration.

Figure 3.17 shows the *number of ids* they assigned, on average, for each browser instance. We see that Panopticlick's algorithm often assigns new browser ids, which is caused by its conservative nature. Indeed, as soon as there is more than one change, or multiple candidates for linking, Panopticlick's algorithm assigns a new id to the unknown browser instance. However, we can observe that both Fp-Stalker's hybrid and rule-based variants perform similarly.

Finally, Figure 3.18 presents the average *ownership* of tracking chains against the collect frequency for the three algorithms. We see that, despite its conservative nature, Panopticlick's

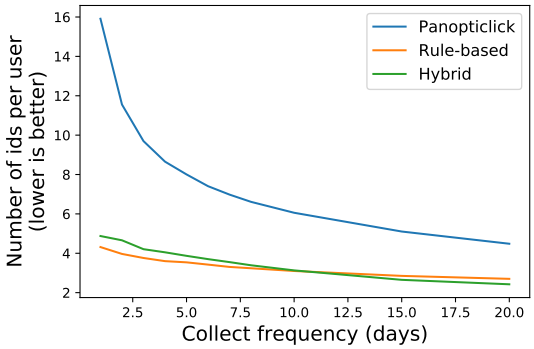


Figure 3.17: Average number of assigned ids per browser instance against simulated collect frequency for the three algorithms (lower is better).

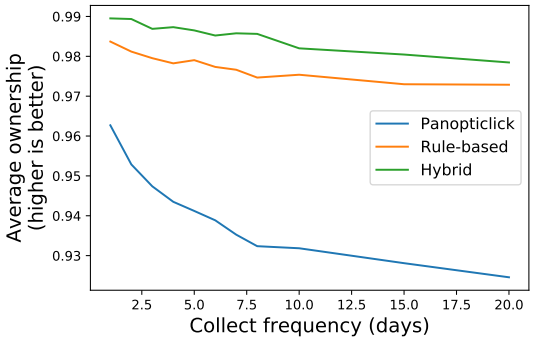


Figure 3.18: Average ownership of tracking chains against simulated collect frequency for the three algorithms. A value of 1 means the tracking chain is constructed perfectly.

ownership is 0.94, which means that, on average, 6% of a tracking chain is constituted of fingerprints that do not belong to the browser instance that owns the chain—*i.e.*, it is contaminated with other fingerprints. The hybrid variant of FP-STALKER has an average *ownership* of 0.985, against 0.977 for the rule-based.

When it comes to linking browser fingerprints, FP-STALKER's hybrid variant is better, or as good as, the rule-based variant. The next paragraphs we focus on a few more results we obtain with the hybrid algorithm. Figure 3.19 presents the cumulative distribution of the average and maximum tracking duration when $collect\_frequency$ equals 7 days for the hybrid variant. We observe that, on average, 15, 5% of the browser instances are tracked more than 100 days. When it comes to the the longest tracking chains, we observe that more than 26% of the browser instances have been tracked at least once for more than 100 days during the experiment. Depending on when the browser entered or left the experiment, most of these browser were tracked for the entire duration of their participation. Few browser instances were present for the whole experiment, most for a few weeks, and at best we can track a browser instance only as long as it was present. These numbers also show how tracking may depend on the browser and its configuration. While some browsers are hardly trackable, others may be tracked for months or likely longer. The graph shows the results of the perfect linking algorithm, which can be interpreted as the distribution of duration of presence of browser instances in our test set.

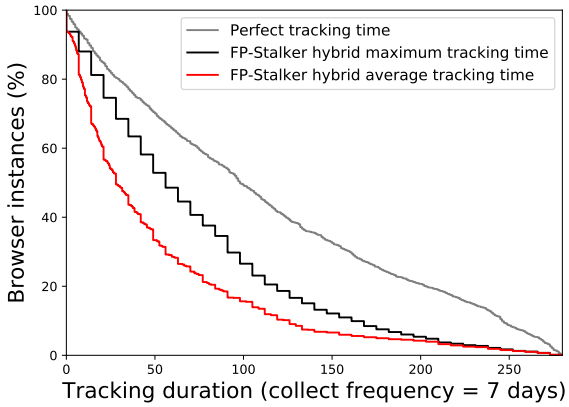


Figure 3.19: CDF of average and maximum tracking duration for a collect frequency of 7 days (FP-STALKER hybrid variant only).

We also look at the distribution of the chains to see how often fingerprints from different

browser instances are mixed together. For the Fp-Stalker hybrid variant, more than 95% of the chains have an ownership superior to 0.8, and more than 90% have perfect ownership—*i.e.*, 1. This shows that a small percentage of browser instances become highly mixed in the chains, while the majority of browser instances are properly linked into clean and relatively long tracking chains.

3.4.3 Discussing Fp-Stalker

We studied browser fingerprint linking, in isolation, which is its worst-case scenario. In practice, browser fingerprinting would be combined with stateful tracking techniques (*e.g.*, cookies, Etags) to respawn stateful identifiers [Acar 2014] (*e.g.*, for advertising), meaning that linking is a much less frequent event for tracking and would primarily serve to extend the lifespan of stateful identifiers, which are being increasing deleted by privacy-aware users. These same users are more likely to have customized their browser configurations, potentially making themselves more unique and linkable through fingerprinting. Another use case is to use browser fingerprinting to enhance authentication [Alaca 2016]. In this case, one only needs to match the fingerprint of the browser attempting to sign-in with the previous fingerprints from the same user, drastically reducing the number of comparisons. The algorithm's strictness can be tuned accordingly.

Our results show that some browser instances have *highly trackable* fingerprints, to the point that very infrequent fingerprinting is quite effective for tracking them. In contrast, other browser instances appear to be *untrackable*. Vendors should work to minimize the attack surfaces exploited by fingerprinters, and users should avoid customizing their browsers in ways that make them expose unique and linkable fingerprints.

3.5 FP-Scanner: Detecting Browser Fingerprint Inconsistencies

A direct consequence of the large variability and diversity of devices, browsers and configurations has lead to browser fingerprint *uniqueness* and *linkability*, which are the main culprits to both the privacy risks and the utility of browser fingerprinting. A range of countermeasures have been proposed to protect against browser fingerprinting, they address either of these properties. Some solutions simply spoof attributes, while others add noise to break the stability required for tracking. However, these solutions, often inadvertently, introduce new information that can be used to further identify devices. Simply knowing of their existence is useful for trackers, and in many cases, further information can be extracted making some solutions counter productive.

This leads us to an interesting property in browser fingerprinting, that of fingerprint *consistency*. Nikiforakis et al. [Nikiforakis 2013] were the first to show that inconsistencies from user agent spoofers could increase fingerprint uniqueness. Many of the attributes in a browser fingerprint can be verified or cross-examined, requiring a coherent approach to defend against identification. Our work in FP-Scanner addresses a wide range of fingerprinting countermeasures. Furthermore, as we have mentioned before, an increase in *uniqueness* does not necessarily translate to an in *linkability*.

We show that most countermeasures negatively impact user privacy. Simply lying about, or badly randomizing, attributes is not a smart idea. We also found fingerprinting scripts in-the-wild that actively detect inconsistencies. We organized a test suite to detect inconsistencies along 4 distinct components, ordered from easiest to most complex they are : operating system, browser, device and canvas. Figure 3.20 shows the 4 our inconsistency test suite. The rest of this subsection is a summarized version of *FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies* [Vastel 2018a] published at Usenix Security 18.

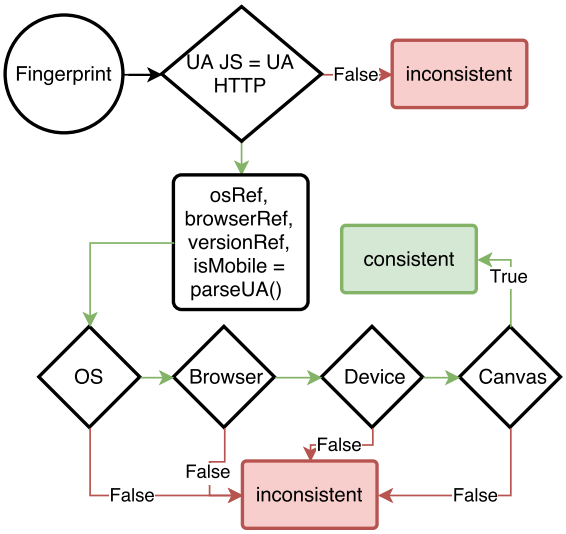


Figure 3.20: Overview of the inconsistency test suite.

3.5.1 Operating System Inconsistencies

Obtaining the browser's user agent or platform is straightforward, but verifying the host OS is more challenging because of the browser's sandbox mechanisms. We present heuristics to verify OS consistency.

User Agent The user agent is available on the client side through the navigator object (`navigator.userAgent`), and on the server side, as an HTTP header (`User-Agent`). We apply checks the equality of these two values. Naive fingerprinting countermeasures, such as basic user agent spoofers, tend to only alter the HTTP header. Differences reflect a coarse-grained inconsistency. While extracting the OS and the browser substrings can help reveal the source of the inconsistency, the similarity of each substring does not necessarily guarantee the OS and the browser values are true, as both might be spoofed. Therefore, we extract and store the OS, browser and version substrings as internal variables `OSref`, `browserRef`, `browserVersionRef` for further investigation.

Navigator platform The value of `navigator.platform` should be consistent with the `OSref`. Consistent does not mean identical, for example, the user agent of a 32-bits Windows contains the substring `WOW64`, which stands for *Windows on Windows 64-bits*, while the attribute `navigator.platform` will report the value `Win32`. Table 3.6 therefore maps `OSref` and possible values of `navigator.platform` for the most commonly used OSes.

WebGL `WebGL` extends the HTML 5 canvas API to render 3D objects. We verify WebGL's `renderer` and `vendor` attributes. `renderer` reports the name of the GPU, for example `ANGLE (VMware SVGA 3D Direct3D11 vs_4_0 ps_4_0)`. The substring `VMware` indicates that the browser is executed in a virtual machine. The `ANGLE` substring stands for *Almost Native Graphics Layer Engine*, which brings OpenGL compatibility to Windows. `vendor` provides the GPU vendor, whose value actually depends on the OS. We summarize the attribute mappings in Table 3.7.

Browser plugins `navigator.plugins` returns a detailed array of information, such as the filename and extension of plugins, which may indicate the OS. On Windows, plugins have `.dll` extensions, while on macOS they are `.plugin` or `.bundle`, and Linux uses `.so`. We test that `OSref` is consistent with plugin filename extensions. Moreover, we also consider constraints imposed by

Table 3.6: Mapping between common `OS` and `platform` values.

OS	Platforms
`Linux`	Linux i686, Linux x86_64, Linux, Linux armv8l
`Windows`	Win32, Win64
`iOS`	iPhone, iPad
`Android`	Linux armv71, Linux i686, Linux armv8l
`macOS`	MacIntel
`FreeBSD`	FreeBSD amd64, FreeBSD i386

Table 3.7: Mapping between common `OSes` and WebGL's `renderer`/`vendor` attributes.

OS	Renderer	Vendor
`Linux`	Mesa, Gallium	Intel, VMWare, X.Org
`Windows`	ANGLE	Microsoft, Google Inc
`iOS`	Apple, PowerVR	Apple, Imagination
`Android`	Adreno, Mali, PowerVR	Qualcomm, ARM, Imagination
`macOS`	OpenGL, Iris	Intel, ATI
`Windows Phone`	Qualcomm, Adreno	Microsoft

some systems, such as mobile browsers that do not support plugins. Thus, reporting plugins on mobile devices is considered an inconsistency.

Media queries Media query is a CSS 3 feature that applies style properties depending on specific conditions. It's most commonly used for responsive web design but it's possible to identify the OS in some cases on Firefox. For example, the Mac graphite theme can be detected using the `-moz-mac-graphite-theme` media query [Takei 2015]. It is also possible to test specific themes present on Windows by using `-moz-windows-theme`. It is also possible to use the `-moz-os-version` media query to detect if a browser runs on Windows XP, Vista, 7, 8 or 10. Moreover, if any of the previous media queries is matched, it likely means that the real browser is Firefox.

Fonts Saito et al. [Saito 2016] demonstrate that fonts may depend on the OS. If a device claims to be on one OS but does not list fonts linked to it, and instead displays fonts from another OS, we consider this to be an inconsistency.

3.5.2 Browser Inconsistencies

Error objects In JavaScript, `Error` objects are thrown when a runtime error occurs. 7 types of errors exist for client-side exceptions. However, for a given error, not all browsers throw the same type of error. For a *stack overflow*, Firefox throws an `InternalError` and Chrome throws a `RangeError`. Instances of the same error type may contain different properties depending on the browser. While two properties—`message` and `name`—are standard, others such as `description`, `lineNumber` or `toSource` are not supported by all browsers. Even for properties, such as `message` and `name`, which are implemented in all major browsers, their values may differ for a given error. For example, executing *null*[0] on Chrome will generate the following error message *"Cannot*

read property '0' of null", while Firefox generates *"null has no properties"*, and Safari *"null is not an object (evaluating 'null[0]')"*.

Functions' internal representations It is possible to obtain a string representation of any object or function in JavaScript by using the `toString` method. However, such representations—*e.g.*, `eval.toString()`—differ depending on the browser, a simple length check can distinguish them. Chrome's message has a length of 33 characters, and 39 for Internet Explorer's, but Firefox and Safari return the same string, with a length of 37 characters The property `navigator.productSub` returns the build number of the current browser. On Safari, Chrome and Opera, it always returns the string `20030107`, not not on Firefox. Combining both, `navigator.productSub` with `eval.toString().length`, can therefore be used to distinguish Firefox from Safari.

Navigator object `Navigator` is a built-in object that represents the state and identity of the browser. Its prototype varies between browser families and versions and is not standardized. For example, the feature `getUserMedia` is available as `mozGetUserMedia` on Firefox and `webkitGetUserMedia` on Webkit-based browser's. Moreover, as `navigator` properties play an important role in browser fingerprinting, our test suite detects if they have been overridden by looking at their internal string representations. In the case of a genuine fingerprint whose attributes have not been overridden in JavaScript, it should contain the substring `native code`. However, if a property has been overridden, it will return the code of the overridden function.

Browser features Browsers are complex and evolve fast. By observing the availability of specific features, particularly in newer versions, it is possible to detect if the browser's `user agent` has been manipulated [Nikiforakis 2013, Mulazzani 2013].

3.5.3 Device Inconsistencies

Browser events Some events are unlikely to happen on a desktop computer, such as touch-related events (`touchstart`, `touchmove`). On the opposite, mouse-related events (`onclick`, `onmousemove`) are unlikely on a smartphone. Therefore, the availability of an event may reveal the device's nature.

Browser sensors Like events, some sensors may have different outputs depending on the nature of the device. For example, the accelerometer, which is generally on mobile devices, can be retrieved from a browser without requesting any authorizations. The value of the acceleration will always slightly deviate from 0 for a real mobile device, even when lying on a flat surface.

3.5.4 Canvas Inconsistencies

Canvas fingerprinting uses the HTML 5 canvas API to draw 2D shapes using JavaScript. This is used to fingerprint browsers [Mowery 2012]. To do so, a sequence of drawing instructions, such as text, shapes or coloring, is sent to the browser and the rendered image is retrieved. Since the rendering relies on the combination of different hardware and software layers, it differs from one device to another. An example of the rendering obtained on a Chrome browser running Linux is presented in Figure 3.21a. We consider these scripted instructions as constraints that must be checked in the rendered image. For example, the canvas in Figure 3.21b has been obtained with the Canvas Defender extension activated. We observe that the background is no longer transparent, which is now a constraint violation. From the rendered image, our test suite checks the following properties:

1. *Number of transparent pixels* as the background of our canvas is transparent.

(a) Canvas fingerprint with no countermeasure.

(b) Canvas fingerprint with a countermeasure.

Figure 3.21: Two examples of canvas fingerprints (a) a genuine canvas fingerprint without any countermeasures installed in the browser and (b) a canvas fingerprint altered by the Canvas Defender countermeasure that applies a uniform noise to all the pixels in the canvas.

2. *Number of isolated pixels*, which are non-transparent pixels surrounded by transparent pixels. These pixels should not occur because we draw closed shapes and texts drawn.

3. *Number of pixels per color*, which is checked against the input canvas rendering script. Even if it is not possible to know in advance the exact number of pixels with a given color, we expect to find colors defined in the canvas script, otherwise we suspect tampering.

We also check if canvas-related functions, such as `toDataUrl`, have been overridden.

Table 3.8: List of attributes collected by our fingerprinting script.

Attribute	Description
`HTTP headers`	List of HTTP headers sent by the browser and their associated value
`User agent navigator`	Value of `navigator.userAgent`
`Platform`	Value of `navigator.platform`
`Plugins`	List of plugins (description, filename, name) obtained by `navigator.plugins`
`ProductSub`	Value of `navigator.productSub`
`Navigator prototype`	String representation of each property and function of the `navigator` object
`Canvas`	Base 64 representation of the image generated by the canvas fingerprint test
`WebGL renderer`	`WebGLRenderingContext.getParameter("renderer")`
`WebGL vendor`	`WebGLRenderingContext.getParameter("vendor")`
`Browser features`	Presence or absence of certain browser features
`Media queries`	Collect if media queries related to the presence of certain OS match or not using `window.matchMedia`
`Errors type 1`	Generate a `TypeError` and store its properties and their values
`Errors type 2`	Generate an error by creating a socket not pointing to an URL and store its string representation
`Stack overflow`	Generate a stack overflow and store the error `name` and `message`
`Eval toString length`	Length of `eval.toString().length`
`Media devices`	Value of `navigator.mediaDevices.enumerateDevices`
`TouchSupport`	Collect the value of `navigator.maxTouchPoints`, store if we can create a `TouchEvent` and if `window` object has the `ontouchstart` property
`Accelerometer`	`true` if the value returned by the accelerometer is not 0, otherwise `false`
`Screen resolution`	Values of `screen.width/ height`, and `screen.availWidth/ Height`
`Fonts`	Font enumeration using JavaScript [Fifield 2015]
`Overwritten properties`	Collect string representation of `screen.width/height` getters, as well as `toDataURL` and `getTimezoneOffset` functions

3.5.5 Empirical Evaluation of FP-Scanner

We implemented tests for all the inconsistencies presented above, including fonts, media queries, overridden functions, device sensors, touch events, screen resolution, canvas poisoning, WebGL,

Table 3.9: Comparison of accuracies per countermeasures

Countermeasure	Number of fingerprints	Accuracy FP-Scanner	Accuracy FP-JS2 / Augur
Random Agent Spoofer (RAS)	69	**1.0**	0.55
User agent spoofers (UAs)	22	**1.0**	0.86
Canvas Defender	26	**1.0**	0.0
Firefox protection	6	**1.0**	0.0
Canvas FP Block	3	**1.0**	0.0
FPRandom	7	**1.0**	0.0
Brave	4	**1.0**	0.0
No countermeasure	10	**1.0**	1.0

exception/error injections, string lengths and overwritten values. We used the browser fingerprint dataset collected from [AmIUnique 2021] to analyze fonts per OS and other attributes, such as the user agent or WebGL's renderer/vendor attributes, in order to build a ground truth values to test for. We also find optimal numbers of fonts, browser features and pixels to test for in order to minimize our false positives. Table 3.8 presents the attributes collected by our fingerprinting script that checks for inconsistencies. Table 3.10 summarizes our results by presenting the set of tests we run, as well as the countermeasures that are detected by each test. The resulting dataset of labeled browser fingerprints and the consistency checks are made available and allow reproducing our results.[2]

Dataset. We created a web page that includes the browser fingerprinting script we designed. We test the countermeasures listed in Table 3.9, as they are representative of the diversity of strategies of current countermeasures. Although other academic countermeasures have been published [FaizKhademi 2015, Laperdrix 2015, Nikiforakis 2015, Torres 2015], it was not possible to consider them due to the unavailability of their code or because they could no longer run. We built our browser fingerprint dataset by accessing the web page from different browsers, virtual machines and smartphones, with and without countermeasures installed. We also collect the system's ground truth—*i.e.*, the real OS, browser and version, as well as the list of countermeasures installed. The resulting dataset is composed of 147 browser fingerprints, randomly challenged by 7 different countermeasures. Table 3.9 reports on the number of browser fingerprints per countermeasure. The number fingerprints is different since some countermeasures are deterministic in the way they operate. For example, Canvas Defender always adds a uniform noise on all the pixels of a canvas. Other countermeasures, such as Random Agent Spoofer, add more randomness due to the usage of real profiles, which requires more tests.

Measuring the accuracy of FP-Scanner. We evaluate the effectiveness of FP-Scanner, FingerprintJS2 [Vasilyev 2019] and Augur [Augur 2018] to correctly classify a browser fingerprint as *genuine* or *altered*. Overall, FP-Scanner reaches an accuracy 1.0 against 0.45 for FingerprintJS2 and Augur, which perform equally on this dataset. When inspecting the Augur and FingerprintJS2 scripts, and despite Augur's obfuscation, we observe that they seem to perform the same tests to detect inconsistencies. As the number of fingerprints per countermeasure is unbalanced, Table 3.9 compares the accuracy achieved per countermeasure. We observe that FP-Stalker outperforms FingerprintJS2 to classify a browser fingerprint as *genuine* or *altered*. In particular, FP-Scanner detects the presence of canvas countermeasures while FingerprintJS2 and `Augur` spotted none of them.

[2]FP-Scanner dataset: https://github.com/Spirals-Team/FP-Scanner

Analyzing the detected countermeasures. For each browser fingerprint, FP-Scanner outputs the result of each test and the value that made the test fail. Thus, it enables us to extract some kinds of signatures for different countermeasures. In this section, we execute FP-Scanner in `depth` mode—*i.e.*, for each fingerprint, FP-Scanner executes all of the steps, even if an inconsistency is detected. For each countermeasure considered in the experiment, we report on the steps that revealed their presence.

User Agent Spoofers are detected by simple consistency checks, such as platform, or a function's internal representation, even when both values of user agent are changed.

Brave is detected through side effects, such as blocking canvas fingerprinting. FP-Scanner detects it overrides `navigator.plugins` and `navigator.mimeTypes` getters, the output is `() => { return handler }`. Moreover, we also detect that Brave overrides `navigator.mediaDevices.enumerateDevices` as it returns a `Proxy` object instead of an object representing the devices.

Random Agent Spoofer (RAS) RAS aims to introduce fewer inconsistencies by using profiles. RAS passes simple checks, such as a user agent consistent with `navigator.platform`. Nevertheless, RAS only ensures consistency between the attributes contained in the profile. Since RAS is a Firefox extension, it is vulnerable to the media query technique. In the case where this is insufficient, plugins or fonts linked to the OS can be used. Browser inconsistencies are also detected, either using functions' internal representations or error attributes. When only the browser version was altered, FP-Scanner detects it by using the combination of Modernizr and Caniuse to check for features. RAS overrides most of the navigator attributes, however, `navigator.vendor` is overridden in JavaScript, which makes it detectable. FP-Scanner also detects devices which claimed to be mobile devices, but whose accelerometer value was undefined.

Firefox fingerprinting protection standardizes the user agent by forcing the value of `Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:52.0) Gecko/20100101 Firefox/52.0`, thus lying about the browser version and the operating system for users not on Windows 7 (Windows NT 6.1). While OS-related attributes, such as `navigator.platform` are updated, other attributes, such as `webgl vendor` and `renderer` are not consistent with the OS. For privacy reasons, since version newer than 57, Firefox disabled OS-related media queries. Nevertheless, when fingerprinting protection is activated, Firefox pretends to be version 52 running on Windows 7. Thus, it should match the media query `-moz-os-version` for Windows 7, which is not the case. Additionally, when the browser was not running on Windows, the list of installed fonts was not consistent.

Canvas poisoners including Canvas Defender, Canvas FP Block and FPRandom were all detected by FP-Scanner. For the first two, as they override canvas-related functions using JavaScript, we detect that the `toDataURL` function has been altered. For all of them, we detect that the canvas pixel constraints were not consistent. Indeed, we did not find enough occurrences of the color $(255, 102, 0, 100)$, but we found pixels with a slightly different color. Moreover, in case of the browser extensions, we also detected an inconsistent number of transparent pixels as they apply noise to all the canvas pixels.

Table 3.10 summarizes, for each countermeasure, the tests that detected inconsistencies. We observe that FP-Scanner succeeds in detecting a wider spectrum of fingerprinting countermeasures (*e.g.*, canvas extensions, FPRandom [Laperdrix 2017] and Brave [Brave 2018]) than Nikiforakis et al.[Nikiforakis 2013].

Recovering the true values. If an inconsistency is detected, FP-Scanner attempts to recover the true value of key attributes, like `OS`, `browser family` and `browser version`. The process is:

- **OS value.** We combine multiple sources of information, including plugins' file extensions, WebGL renderer, media queries, and fonts linked to OS. For each step, we obtain a possible OS. We select the OS that has been most predicted.

- **Browser family.** We rely on `eval`'s internal representation, specifically it's length (`eval.toString().length`), that we combine with the value of `productSub`. Since these

Table 3.10: FP-Scanner steps failed by countermeasures

Test (scope)	*RAS*	*UA spoofers*	*Canvas extensions*	*FPRandom*	*Brave*	*Firefox*
`User Agents` (global)						
`Platform` (OS)		✓				
`WebGL` (OS)	✓	✓				✓
`Plugins` (OS)	✓	✓				
`Media Queries (OS, browser)`	✓	✓				✓
`Fonts` (OS)	✓					✓
`Error` (browser)	✓	✓				
`Function representation (browser)`	✓					
`Product` (browser)	✓	✓				
`Navigator` (browser)	✓				✓	
`Enumerate devices (browser)`					✓	
`Features` (browser)	✓	✓				
`Events` (device)	✓	✓				
`Sensors` (device)	✓	✓				
`ToDataURL` (canvas)	✓		✓			
`Pixels` (canvas)			✓	✓	✓	✓

two attributes are discriminatory enough to distinguish most major browsers, we do not make more tests.

- **Browser version.** To infer the browser version, we test the presence or absence of each Modernizr feature for the recovered browser family. Then, for each browser version, we count the number of detected features. Finally, we keep a list of versions with the maximum number of features in common.

We applied this algorithm to fingerprints altered only by countermeasures that change the OS or the browser—*i.e.*, RAS, *User agent spoofers* and Firefox fingerprinting protection. FP-Scanner was able to correctly recover the browser ground value for 100 % of the devices. Regarding the OS, FP-Scanner was always capable of predicting the OS family—*i.e.*, Linux, MacOS, Windows—but often failed to recover the correct version of Windows, as the technique we use to detect the version of Windows relies on Mozilla media queries, which stopped working after version 58. Finally, FP-Scanner failed to faithfully recover the browser version. Given the lack of discriminatory features in Modernizr, FP-Scanner can only recover a range of candidate versions.

3.5.6 Discussing FP-Scanner

Discrimination. Being detected with a countermeasure could lead to discrimination. For example, Hannak et al. [Hannak 2014] show that some websites adjust prices depending on the user agent. Moreover, many websites refuse to serve browsers with ad blockers or users of the Tor browser and network. We can imagine users being delivered altered content or being denied access if they do not share their true browser fingerprint.

Tracking. Detecting countermeasures can, in some cases, be used to improve fingerprint linkability. Nikiforakis et al. [Nikiforakis 2013] talk about the counterproductiveness of user agent spoofers because they make browsers more identifiable. We extend this line of thought to more generally argue that being detected with a fingerprinting countermeasure can make browsers more trackable, albeit this is not always the case. We assert that the ease of tracking depends on different factors, such as being able to identify the countermeasure, the number of users of the

countermeasure, the ability to recover the real fingerprint values, and the volume of information leaked by the countermeasure. To support this claim, we provide the following analysis.

Anonymity set. In the case of countermeasures with large user bases, like Firefox with fingerprinting protection or Brave, although their presence can be detected, these countermeasures tend to increase the anonymity set of their users by blocking or standardizing attributes. Since they are used by millions of users, the information obtained does not compensate the loss in entropy from the removal of fingerprinting attributes. Nevertheless, for countermeasures with small user bases, such as Canvas Defender ($21k$ downloads on Chrome, $5k$ on Firefox) or RAS ($160k$ downloads on Firefox), it is unlikely that the anonymity gained by the countermeasures compensates the information obtained by knowing that someone uses them.

Increasing uniqueness. In the case of RAS, we show that it is possible to detect its presence and recover the original browser and OS family. Also, since the canvas attribute has been shown to have high entropy, and RAS does not randomize it nor block it by default, the combination of few attributes of a fingerprint may be enough to identify a RAS user. Thus, under the hypothesis that few RAS users have the same canvas, many of them could be identified by looking at the following subset of attributes: `being a RAS user`, `predicted browser`, `predicted OS`, and `canvas`.

Blurring noise. In the case of Canvas Defender, even though they claim to have a safer solution than other canvas countermeasure extensions, the way they operate makes it easier for a fingerprinter to track their users. Indeed, Canvas Defender applies a uniform noise vector on all pixels of a canvas. This vector is composed of 4 random numbers between -10 and 30 corresponding to the red, green, blue and alpha (`rgba`) components of a color. With a small user base, it is unlikely that two or more users share both the same noise and the same original canvas. In particular, the formula hereafter represents the probability that two or more users of Canvas Defender among k share the same noise vector, which is similar to the birthday paradox: $1 - \prod_{i=1}^{k}(1 - \frac{1}{40^4 - i})$. Thus, if we consider that the $21k$ Chrome users are still active, there is a probability of 0.0082 that at least two users share the same noise vector. Moreover, by default Canvas Defender does not change the noise vector. It requires the user to trigger it, which means that if users do not change the default settings or do not click update the noise vector, they may keep the same noise vector for a long period. Thus, when detecting that a browser has Canvas Defender installed, which can be easily detected as the string representation of the `toDataURL` function leaks its code, if the fingerprinting algorithm encounters different fingerprints with the same canvas value, it can conclude that they originate from the same browser with high confidence. In particular, we discovered that Canvas Defender injects a script element in the DOM (cf. Listing 3.1). This script contains a function to override canvas-related functions and takes the noise vector as a parameter, which is not updated by default and has a high probability to be unique among Canvas Defender users. By using the JavaScript Mutation observer API [Docs 2018] and a regular expression (cf. Listing 3.2), it is possible to extract the noise vector associated to the browser, which can then be used as an additional fingerprinting attribute.

```
1   function overrideMethods(docId, data) {
2     const s = document.createElement('script')
3     s.id = getRandomString();
4     s.type = "text/javascript";
5     const code = document.createTextNode('try{('+overrideDefaultMethods+')(' +data.r+ ','+data.g
      ↪  + ',' + data.b + ','+data.a+',"'+s.id
      ↪  +'","'+storedObjectPrefix+'");}catch(e){console.error(e);}');
6     s.appendChild(code);
7     var node = document.documentElement;
8     node.insertBefore(s, node.firstChild);
9     node[docId] = getRandomString();
10  }
```

Source Code 3.1: Script injected by Canvas Defender to override canvas-related functions

```
1   var o = new MutationObserver((ms) => {
2     ms.forEach((m) => {
```

```
3         var script = "overrideDefaultMethods";
4         if (m.addedNodes[0].text.indexOf(script) > -1) {
5           var noise = m.addedNodes[0].text.match(/\d{1,2},\d{1,2},\d{1,2},\d{1,2}/)[0].split(",");
6         }
7       });
8     });
9     o.observe(document.documentElement, {childList:true, subtree:true});
```

Source Code 3.2: Script to extract the noise vector injected by Canvas Defender

Protection level. While it may seem more tempting to install an aggressive fingerprinting countermeasure—*i.e.*, a countermeasure, like RAS, that blocks or modifies a wide range of attributes used in fingerprinting—we believe it may be wiser to use a countermeasure with a large user base even though it does not modify many fingerprinting attributes. Moreover, in the case of widely-used open source projects, this may lead to a code base being audited more regularly than less adopted proprietary extensions. We also argue that all the users of a given countermeasure should adopt the same defense strategy. Indeed, if a countermeasure can be configured, it may be possible to infer the settings chosen by a user by detecting side effects, which may be used to target a subset of users that have a less common combination of settings. Finally, we recommend a defense strategy that either consists in blocking the access to an attribute or unifying the value returned for all the users, rather than a strategy that randomizes the value returned based on the original value. Concretely, if the value results from a randomization process based the original value, as does Canvas Defender, it may be possible to infer information on the original value.

3.6 Summary

In this chapter we presented our work around fingerprinting through three large-scale empirical studies to understand the main properties that make browser fingerprinting a risk to privacy, useful for security, and otherwise functional. We are referring to the *uniqueness*, *linkability* and *consistency* of browser fingerprints.

In *Beauty and the Beast*, we performed the second large scale study of browser fingerprint uniqueness, six years after Eckersley's paper in 2010 [Eckersley 2010]. We confirmed that browser fingerprinting was still viable, but more importantly, we showed that modern technologies being implemented, namely HTML5, posed a risk because they were increasing the browser's attack surface. We no longer needed plugins to fingerprint devices, the browser now provides hardware accelerated APIs that give detailed information. More surprisingly, through the first large scale study of mobile device fingerprints, we found similar levels of uniqueness despite our understanding of browser fingerprinting attributes in desktops at the time. Indeed, through customized fonts, emojis, hardware-based APIs such as the HTML5 canvas, and overly detailed and customized user agents, we found most devices to exhibit unique browser fingerprints.

In Fp-Stalker, we implemented two variants of linking algorithms to re-identify browser instances, and we ran different scenarios to test their effectiveness. The first variant builds on a ruleset identified from an analysis of grounded programmer knowledge. The second variant combines the most discriminating rules and leverages machine learning to sort out the more subtle ones. We trained the Fp-Stalker hybrid variant with a training set of fingerprints that we collected for 2 years through browser extensions installed by $1,905$ volunteers. Our experiments demonstrate that the hybrid variant can correctly link fingerprint evolutions from a given browser instance for 54.48 consecutive days on average, against $42,3$ days for the rule-based variant. When it comes to the maximum tracking duration, with the hybrid variant, more than 26% of the browsers can be tracked for more than 100 days. In many cases, we find that the duration of their participation in the dataset becomes the limiting factor. That is, we find these browsers to be very trackable. We argue that linkability and fingerprint tracking, in a real world scenario, would likely be limited to respawning of stateful identifiers, drastically reducing the overhead of the fingerprint linking algorithms and allowing trackers to focus on users that delete their cookies but (over-)customize their browsers and devices.

In FP-Scanner, we identified attributes to detect inconsistencies. Thus, instead of taking the value of a fingerprint for granted, fingerprinters could check whether attributes of a fingerprint have been modified to escape tracking algorithms, and apply different heuristics accordingly. We collected 147 browser fingerprints extracted from browsers using state-of-the-art fingerprinting countermeasures and we showed that FP-Scanner was capable of accurately distinguishing genuine from altered fingerprints. We discussed how detecting fingerprinting countermeasures, as well as being capable of predicting the true values of spoofed attributes, may impact privacy. We argued that being detected with a fingerprinting countermeasure does not necessarily imply being tracked more easily. Being trackable depends on the capability of identifying the countermeasure used, the number of users having the countermeasure, the capacity to recover the original fingerprint values, and the information leaked by the countermeasure. Our work shows that proper countermeasures must be designed and implemented diligently or they risk being counterproductive through the detection of inconsistencies that increase uniqueness and linkability.

Doctoral student supervision associated to this chapter

Antoine VASTEL, 2016-2019 (66%, co-advised with Romain Rouvoy)
Tracking Versus Security: Investigating the Two Facets of Browser Fingerprinting.
Doctoral contract, Université de Lille
Defended October 2019. Currently working at Datadome in bot detection.

Antonin DUREY, 2018-2021 (50%, co-advised with Romain Rouvoy & Lionel Seinturier)
Leveraging Browser Fingerprinting to Fight Fraud on the Web.
Financed through a collaboration with the French Army Ministry.

Publications associated with this chapter

2021 Antonin Durey, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **FP-Redemption: Studying Browser Fingerprinting Adoption for the Sake of Web Security.** The 18th Conference on Detection of Intrusions and Malware & Vulnerability Assessment - DIMVA'21, Jul 2021, Lisboa, Portugal. https://hal.inria.fr/hal-03212726

2021 Antonin Durey, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **An iterative technique to identify browser fingerprinting scripts. 2021.** https://hal.inria.fr/hal-03212729

2020 Sarah Bird and Vikas Mishra and Steven Englehardt and Rob Willoughby and David Zeber and Walter Rudametkin and Martin Lopatka, **Actions speak louder than words: Semi-supervised learning for browser fingerprinting detection**, 2020. https://arxiv.org/abs/2003.04463

2018 Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **FP-STALKER: Tracking Browser Fingerprint Evolutions.** 39th IEEE Symposium on Security and Privacy (S&P 2018). https://hal.inria.fr/hal-01652021
Core Rank: A / Google Scholar (Computer Security): #2 – Acceptance rate: 11%.*

2018 Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies.** 27th USENIX Security Symposium (USENIX Sec. 2018, Baltimore, USA). https://hal.inria.fr/hal-01820197
Core Rank: A/ Google Scholar (Computer Security): #4 – Acceptance rate: 19%.*

2016 Pierre Laperdrix, Walter Rudametkin, Benoit Baudry. **Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints.** 37th IEEE Symposium on Security and Privacy (S&P 2016). *Rank: A* / Google Scholar (Computer Security): #2 – Acceptance rate: 13%.*
⧗ **2018 CNIL-Inria European Privacy Protection Award.**

Research grants associated with this chapter

2018 Research and collaboration contract with *Ministère des Armées* on the topic of **Leveraging Browser Fingerprinting to Fight Fraud on the Web**. Antonin Durey is funded under this contract. Total funding: 163k€.

2018 Inria ADT (*Action de Développement Technologique*) funding. The **FingerKit: a Cloud Platform to Study Browser Fingerprints at Large** project served to finance 24 engineer-months on a full rebuild of the AmIUnique platform with new functionality.

2016 **PEPS JCJC INS2I FPDefendor**. Project to explore techniques to protect users against browser fingerprint tracking. Total funding: 8700€.

2016 Université de Lille doctoral grant on the subject of **Machine learning to optimize privacy against browser fingerprint tracking**. This grant financed Antoine Vastel's doctoral contract.

Chapter 4

Breaking Linkability, Reducing Uniqueness, and Exploiting Consistency

Contents

4.1 Motivations . . . 51
4.2 Overview . . . 52
4.3 Blink: Breaking *Linkability* Through Multi-Level Reconfiguration . . . 53
4.3.1 A moving target defense against tracking . . . 53
4.3.1.1 Balancing privacy and browsing comfort . . . 55
4.3.1.2 Dissimilarity metric . . . 55
4.3.1.3 Comparing Blink . . . 56
4.3.2 Multi-level reconfiguration . . . 57
4.3.2.1 The diversity reservoir . . . 58
4.3.2.2 Modeling the well-formedness constraints in the diversity reservoir 58
4.3.3 Empirical validation of multi-level reconfiguration . . . 59
4.3.4 Discussing Blink . . . 60
4.4 Fp-Tester: Automated Testing to Reduce Fingerprint Uniqueness . . . 61
4.4.1 Short-Term Fingerprintability of Browser Extensions . . . 61
4.4.2 Long-Term Fingerprintability of Browser Extensions . . . 62
4.4.3 Fingerprintability Reports and Components . . . 63
4.4.4 Preliminary Results: Random Agent Spoofer . . . 64
4.4.5 Discussing Fp-Tester . . . 65
4.5 FP-Crawlers: exploiting fingerprint consistency to block crawlers . . . 66
4.5.1 Websites that block crawlers with fingerprinting . . . 66
4.5.2 Characterizing fingerprinting scripts . . . 67
4.5.3 Effectiveness of browser fingerprinting to detect crawlers . . . 68
4.5.4 Discussing FP-Crawlers . . . 71
4.6 Summary . . . 71

4.1 Motivations

Browser fingerprint *uniqueness* and stability lead to *linkability*, and browser fingerprinting *consistency* establishes how difficult it is to spoof or overcome. When browser fingerprinting was just beginning, *consistency* was arguably the most important reason the security community were quite pessimistic about fingerprinting and felt it was going to be a complex task to address. Nobody expects to force people to use identical software and hardware, and given the numerous ways to verify these, meant attackers could always find a way to reidentify and track devices. This chapter focuses on three of our works that attempt to address, or exploit, each of the properties we presented in Chapter 3.

We have always been worried about the risks of browser fingerprint tracking, and the potential for browser fingerprinting to be a transparent, server-side controlled, stateless, insidious replacement for cookies. One of the only privacy advantages of cookies are that they are controlled by the client's device. Browser fingerprinting changes that. So in 2014, with our software engineering backgrounds, and our experience in component-based software engineering and dynamic reconfiguration, we figured that it was possible to extensively randomize a browsing environment, forcing the user's browser to exhibit a different fingerprint every time because it's environment was different each time. We were pushing against the consensus at the time, which was to reduce diversity, choice and configuration to make browser fingerprints identical. Our idea was simple, if between two browsing sessions my fingerprint is so different as to be unlinkable, then it's also untrackable. We also figured we could do so in a usable manner with acceptable performance. This led to our first work in browser fingerprinting. Our objective was to **break the linkability** of browser fingerprints through random reconfigurations using a large diversity pool of software components. In essence, exploiting the same diversity that was introducing the risks to privacy. In fact, at the time, it was mostly our intuition that led us to think that attackers would always find ways to identify spoofing or less ambitious randomization approaches. Of course, as shown by FP-Scanner in Section 3.5, we later found our intuition to be pretty spot-on.

With more experience, specifically running `https://amiunique` and after publishing *Beauty and the Beast*, FP-Stalker and FP-Scanner, we found that a lot of the attributes and side-effects being made available to fingerprinters, by developers, could, in many cases, be avoided. In essence, we figured that some assistance during the development phases that could give insight on the fingerprintability of their solutions, could help developers and vendors to **reduce uniqueness**. This lead is to FP-Tester, an introductory study to understand the feasibility of automated fingerprint analyses during the development of browsers and browser extensions.

Finally, we became ever more interested in how fingerprinting was being used in-the-wild. Our understanding of the *consistency* of browser fingerprints led us to believe that fingerprinting was being used for security measures. We identified Web crawlers as a prime suspect and, in FP-Crawlers, we studied how **fingerprint consistency is being exploited** in-the-wild.

4.2 Overview

Breaking linkability. We capitalize on the existing diversity of client-side software to *break fingerprint linkability* through constant random reconfigurations at multiple levels. A moving target defense strategy against browser fingerprint tracking. We chose to randomize four components, which at the time represented the main sources of uniqueness in a browser fingerprint: browsers, plugins, fonts, and the operating system. Our paper *Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification*, published at SEAMS 2015, introduces the tool we developed, called *Blink*, which stands for "**b**reaking **link**ability". The main contributions of this paper are:

- A moving target defense that leverages software diversity and dynamic reconfiguration to automatically assemble diverse browsing platforms.
- A dissimilarity metric that estimates the likelihood that a fingerprinter will consider two fingerprints as coming from the same platform.
- An experimental assessment of Blink's ability to automatically assemble random, dissimilar, and valid platforms.

Reducing uniqueness. Seeing how sensitive fingerprint *uniqueness* is, and the results of developer's inadvertently making browser fingerprints easier to track, we produced a toolkit to automatically test and analyze browser fingerprints. We chose to focus on fingerprinting countermeasures because we have already shown they are often detectable and sometimes counterproductive to user's privacy because they introduce inconsistencies (see 3.5). Our toolkit, FP-Scanner, makes the following contributions;

- Proposes a series of inconsistency tests to characterize the short-term fingerprintability of a browser extension.
- Proposes a technique to simulate the use of browser extension fingerprint modifications over large amounts of real fingerprints in order to characterize the long-term fingerprintability of a browser extension.
- Test multiple browser fingerprinting countermeasures and identifies the inconsistencies they produce.

Exploiting consistency. We perform a study to understand how browser fingerprinting is being used in-the-wild, specifically to block crawlers. Because crawlers are ubiquitous, and many websites attempt to block them from scraping their data, our intuition was that fingerprinting would be used to do so. In our paper, FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers, published at MADWeb 2020 and given the best paper award, we find that browser fingerprinting is effective and rapid in blocking crawlers, but, by itself, it is no match to a determined adversary. The contributions of our work are

- We crawl Alexa's Top 10K and identify websites that use browser fingerprinting to block crawlers and analyze the scripts.
- We deobfuscate the 4 most common fingerprinting scripts that detect crawlers and present key attributes and how they reveal crawlers.
- We create multiple crawlers with distinct fingerprints to evaluate the resilience of fingerprinting against adversaries that try to evade detection. We measure the effort required and show that fingerprinting is good at detecting crawlers that evade state-of-the-art techniques, all major fingerprinting scripts test *inconsistencies* to identify crawlers, but they can still be bypassed by knowledgeable adversaries.

4.3 Blink: Breaking Fingerprint *Linkability* Through Multi-Level Reconfiguration

We argue that the same diversity that leads to browser fingerprint *uniqueness*, combined with multi-level software reconfiguration, provides the foundations for a counter measure to browser fingerprint tracking. The idea is as follows. A browsing platform is assembled by randomly selecting a coherent set of components (an OS, a browser, plugins, etc.). Third parties collect, via a Web script, enough information about the platform's components to form a fingerprint. We then regularly reconfigure the platform—thanks to a large set of diverse components—causing a different fingerprint to be exhibited. Our approach breaks one essential property for the exploitation of browser fingerprints: their **linkability**.

Blink, short for *breaking linkability*, is our implementation of a *moving target defense against browser fingerprint tracking*. We aim at regularly changing what were the four most distinctive elements in a fingerprint at the time: the operating system, the browser, the list of fonts and the list of plugins. We call these the *Diversified Platform Components*, or DPC. Our approach consists in creating DPC configurations by randomly selecting components from a large pool, called the diversity reservoir. The DPC configurations are then used to assemble and reconfigure the browsing platforms, leading to diverse fingerprints being exhibited over time. The rest of this section is a summarized version of the paper *Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification* published at SEAMS in 2015 [Laperdrix 2015] and describes our approach, implementation and experiments.

4.3.1 A moving target defense against tracking

We propose to automatically reconfigure a user's platform to exhibit different fingerprints over time that cannot easily be linked to one another. Figure 4.1 shows the elements of a browsing platform that affect the fingerprint: configuration data at different levels (HW, OS, browser);

software components that are assembled at different levels (e.g., `apt-get`, browser plugins, fonts); hardware components, such as the graphics card; cross-level dynamic attributes collectable only at runtime, such as through the HTML5 canvas. Once a user starts browsing the web, these data are used to create a *fingerprint*. We say that a platform *exhibits* said fingerprint.

Our approach reconfigures components that affect the exhibited fingerprint. At the time, Eckersley [Eckersley 2010] found the most distinguishing attributes of a fingerprint to be fonts, plugins and user agents. Other known attributes are often shared by a large pool of users, rendering them less discriminating. Based on this, we identified the following set of **Diversified Platform Components (DPC)** to be automatically reconfigured: fonts, plugins, browsers, the operating system and the CPU architecture. We call a **DPC configuration** a consistent assembly of a browser running in an operating system, with a specific set of fonts and plugins.

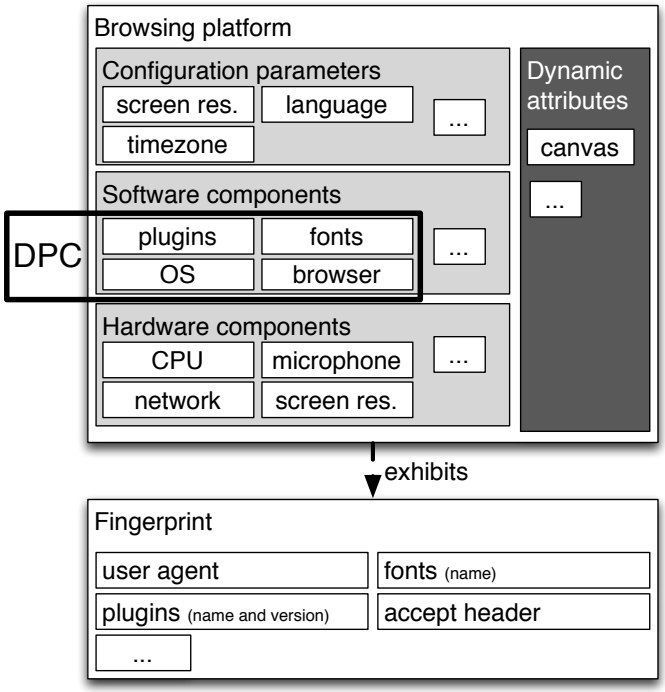


Figure 4.1: Browsing platform elements exhibited in the browser fingerprint.

Definition 1 ***Diversity reservoir*** *is the set of components used to assemble new configurations of the DPC. Given a set O of operating systems for different architectures, a set B of browsers of different types and versions, a set F of fonts and a set P of plugins, the reservoir is $DR = O \cup B \cup F \cup P$.*

Our intuition was that the same set of diverse software components that cause fingerprint uniqueness can be exploited to create very large diversification reservoirs. These can be used to build trillions of distinct configurations to switch among. Figure 4.2 illustrates this principle: we randomly select components from the diversity reservoir to create DPC configurations used in the browsing platforms. Over time we generate new configurations that replace previous ones, changing the browsing platforms and the exhibited fingerprints. The user decides when these changes occur. This approach falls into the family of dynamic platforms, a specific form of moving target approaches, as described by Okhravi et al. [Okhravi 2014]. Our approach is characterized by three essential properties: (i) the assembled platforms always exhibit *consistent fingerprints* because the platforms are **genuine** and we do not lie about any attributes; (ii) we assemble *correct platforms*, i.e., platforms composed of compatible components which run correctly; and (iii) each *reconfiguration causes the exhibited fingerprints to change.*

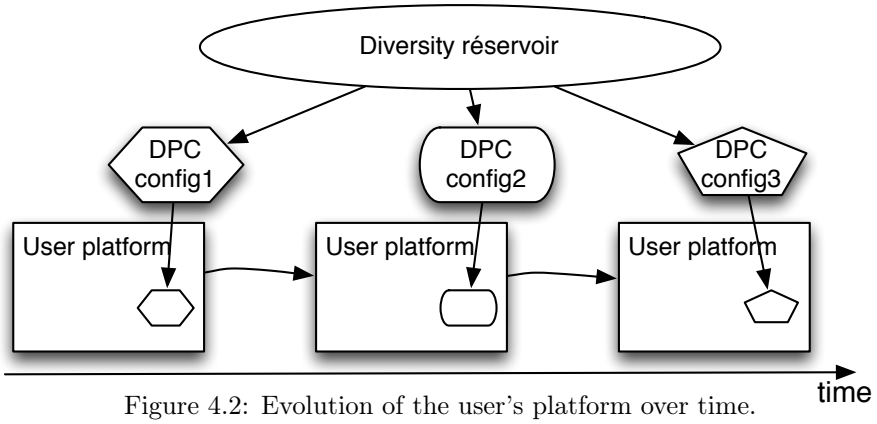


Figure 4.2: Evolution of the user's platform over time.

4.3.1.1 Balancing privacy and browsing comfort

There are many places where user comfort conflicts with potential gains in privacy. For example, we have included browsers in the DPC because it is an important distinguishing factor in a fingerprint. Yet, users populate their browsers with data such as bookmarks, passwords and open tabs. We believe that it is important to transfer user data between platforms we assemble in order to keep a comfortable browsing experience despite switching DPC configurations. We developed an offline cross-browser tool that is responsible for transferring the user profile between browsing platforms. We also acknowledge that not all users are ready to randomize their favorite browsers to obtain a different fingerprint, so we let them customize DPC components. Users select components from the DPC that will be included or excluded from all configurations. We call **alterable** and **essential** the components that a user decides to include or discard from the set of DPC.

Deciding when to reconfigure can also impact privacy and comfort. Frequent changes can be effective at mitigating tracking, but are disturbing to users. However, waiting too long increases the time a user's fingerprint remains static, leading to larger sessions that can be tracked. We provide two strategies for reconfiguration, a full reconfiguration strategy, called *Leery*, and a light-weight one, called *Coffee break*.

Definition 2 *The **Leery strategy** reconfigures all levels of the browsing platform: the operating system, the browser, the fonts and the plugins. Essential components are kept and alterables are randomized. This strategy is used each time a user starts a new session (i.e. starts a browser for the first time). A fresh DPC configuration is generated and kept until the user ends the session. It draws its name from its cautiousness by reconfiguring as many components as possible.*

Definition 3 *The **"Coffee break" strategy** aims to be faster than Leery by reconfiguring only fonts and plugins. The browser and the operating system do not change. As always, essential components are kept and alterable components are randomized. This strategy is triggered in different ways: manually, by having the user lock his computer, or by waiting for a period of inactivity, hence the name.*

It should be noted that reconfigurations do not occur while the user is browsing the web. To explicitly change the current fingerprint, a new session should be started (e.g., by restarting Blink) or a Coffee break reconfiguration should be triggered. Other strategies are also possible, but we feel these two cover a wide range of uses. Our experiments show that both strategies are effective.

4.3.1.2 Dissimilarity metric

To assess Blink's effectiveness, we needed to quantify the differences between fingerprints. To our knowledge, no established measure to compare two fingerprints or to determine if two different

Table 4.1: Changed attributes for example n°1

Attribute	Modified value
User agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, Gecko) Chrome/34.0.1847.116 Safari/537.36
Plugins	Plugin 0: Chrome PDF Viewer; libpdf.so, [...] Plugin 12: gecko-mediaplayer 1.0.9;

Table 4.2: Changed attributes for example n°2

Attribute	Modified value
User agent	Mozilla/5.0 (X11; Linux i686; rv:26.0) Gecko/20100101 Firefox/26.0

fingerprints can be related to a single platform existed. We defined the following dissimilarity metric that aims at capturing the differences between the attribute values observed in two fingerprints.

$$D(FP_1, FP_2) = \frac{\sum_{i=1}^{8} \omega_{attr_i} \times d(attr_i(FP_1), attr_i(FP_2))}{\sum_{i=1}^{8} \omega_{attr_i}}$$

The metric is the sum of dissimilarities between 8 attributes. Its value is defined in the range [0,1]. Each attribute is weighted according to Eckersley's study [Eckersley 2010]. Heavier weights indicate more revealing attributes. This captures, for example, that two fingerprints with the same fonts are closer than two fingerprints with the same timezone. The specific weights ω we used, and the dissimilarity functions d, are defined in the paper.

To illustrate the intuition behind our metric, if we change the browser from Firefox to Chrome and add plugins (as shown in Table 4.1), the dissimilarity between the new and original fingerprints is 0.46. Intuitively, a third party would hardly consider fingerprints with different browsers and plugins to come from the same user. In the second example, only the browser version is modified (Table 4.2). The dissimilarity is then 0.01, which indicates that the two fingerprints are almost identical. This fits our intuition: it is very likely that a browser's version will change at some point in time (e.g., it is updated). Thus, similar fingerprints according to our metric are fingerprints that are likely to be detected as originating from the same user. These two examples illustrate that some differences are more revealing than others, which is what our dissimilarity metric is capturing.

4.3.1.3 Comparing Blink

Our approach is to reconfigure the browsing platform at multiple levels, causing the platform to exhibit a different fingerprint each time. Blink's advantage over spoofers is to never lie. Fingerprints exhibited by the browsing platforms are based on genuine configurations, with genuine browsers, and genuine operating systems. By construction, there are no inconsistencies among the fingerprint attributes, which prevents standing out as a liar. Blink relies on a secretless strategy because knowing the fingerprint reconfiguration strategy (random selection of DPC components) is not enough to defeat it. Cox et al. give insight into the wealth of security advantages obtained through software diversity, particularly when there are no secrets to hide [Cox 2006]. An important advantage is that this Blink is not susceptible to the inconsistencies presented in FP-Scanner(see Section 3.5, and is also resistant to cross-browser fingerprinting, which has shown that simply changing browsers is not enough because other attributes, such as fonts and plugins, are sufficient to create unique fingerprints [Boda 2012].

The main drawbacks of the Tor browser are its usability and the brittleness of its fingerprint.

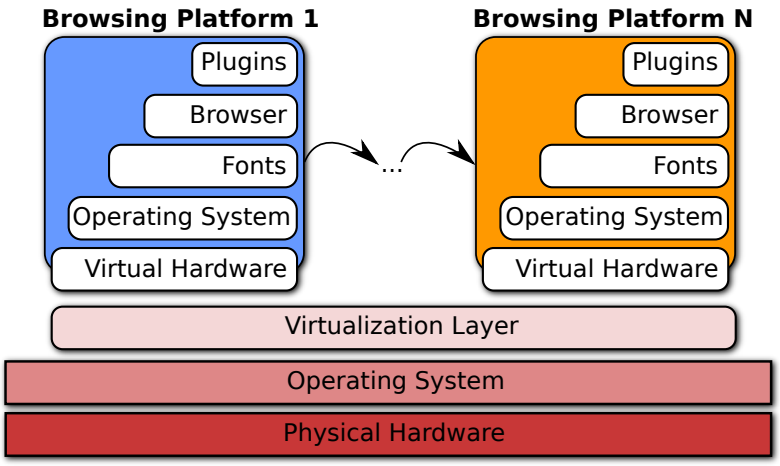


Figure 4.3: A multi-level view of browsing platforms. Virtualization isolates the user's system.

Users should not change any of the Tor browser's configuration options, nor add plugins, because their fingerprint will diverge from the base fingerprint, disrupting the *unique fingerprint* approach and making them identifiable. Our approach is the opposite: we create unstable, always changing platforms that exhibit very different fingerprints. We welcome the addition of user plugins and fonts. And we do not force the use of any restrictive extensions (e.g., NoScript) that severely alter the browsing experience.

Finally, we do not rely on blocking fingerprinting scripts. Maintaining proper lists of scripts requires significant effort. Blink works as a moving target defense system that is oblivious to fingerprinting scripts. Blink also has the potential to resist currently unknown fingerprinting attacks thanks to the use of multi-level reconfigurations.

4.3.2 Multi-level reconfiguration

Although it is possible to randomize some components directly in the user's host system, changes such as adding and removing fonts or plugins can have negative side-effects on other applications. More importantly, certain components, such as the operating system or the CPU architecture, cannot be directly changed. For these reasons, Blink uses virtual machines to maximize the diversity space that can be exploited by randomizing the operating system and potentially the CPU architecture, all the while maintaining a high degree of isolation between the user's system, fingerprinters and the reconfiguration process. Figure 4.3 presents our *multi-level reconfiguration*.

Operating system reconfiguration Blink uses VirtualBox to isolate browsing platforms because it is open source and multi-platform (supports Linux, Mac OS and Windows as both host and guest). Also, VirtualBox abstracts hardware (enabling us to randomize the CPU architecture or graphics card configurations), provides extensive functionality, has low overhead, and allows for a number of configuration parameters. Added bonuses are a user-friendly GUI and guest additions to improve integration. However, it could be substituted with other solutions.

We use VirtualBox's shared folders to transfer data between the host and the VMs. Launching a browsing platform starts the assembly of a randomized DPC configuration (i.e., OS, fonts, plugins, browser). The browser, fonts and plugins are copied to shared folders, as is the user's profile (i.e., open tabs, bookmarks, downloads, saved passwords). VMs start the `BlinkVM.py` monitoring script, whose job is to install fonts and plugins, prepare the user profile, launch and monitor the browser, and listen for commands from the host (e.g., shutdown, coffee break reconfiguration). When shutting down, shared folders are cleaned and the user-profile is recovered. VMs are additionally snapshotted when launched and rolled back to their pre-browsing states

when the browsing session ends. This ensures the platform returns to a known stable state that is free of temporary files, cookies, trackers, and malware.

Font reconfiguration To install a font, the font file must be copied to the correct folder and registered. However, not all browsers detect font changes dynamically and may require restarting.

Browser reconfiguration Reconfiguring a browser requires copying the installation folder and running the main executable.

Plugin reconfiguration Installing a plugin is as simple as copying a file to the proper plugin directory. We focused on randomizing NPAPI plugins (compatible with most browsers) and PPAPI plugins for Chrome (as of January 2015, NPAPI plugins are blocked by default [1]). When reconfiguring, plugins must match the CPU architecture (e.g., i686, IA-64, amd64), the operating system (e.g., Windows, Linux), and the browser (e.g., Firefox, Chrome) of the DPC to work properly. Interestingly, plugin support is often dynamic, as is the case with Chrome, Opera and Firefox. This allows Blink to reconfigure the plugin list at runtime, and is used for Coffee break reconfigurations to immediately change the exhibited fingerprint without restarting the browsing platform.

4.3.2.1 The diversity reservoir

The operating system is the base of the browsing platform. We built VMs using using Ubuntu and Fedora Linuxes for both the i686 and x86_64 architectures. Once built, each VM must be configured to work with Blink. This requires setting up shared folders and configuring Blink's monitoring script to auto-start.

We focused on the Chrome and Firefox browsers and downloaded multiple versions of each from the vendors' websites. There's one version for each combination of release channel (stable, beta and development) and CPU architecture (i686 and x86_64). It is easy to add browsers and there are a number of forks for both Chrome (Chromium, SRWare Iron, Epic, ...) and Firefox (Iceweasel, Seamonkey, ...) that are trivial to integrate. To add a browser you must unpackage it into Blink's browser folder and follow the naming convention.

We wrote scripts to crawl the Ubuntu and Fedora repositories for fonts. We found many duplicate fonts, yet some fonts are specific to a distribution. Moreover, fonts are most often installed in groups, as seen with packages that install multiple fonts. Blink allows defining font groups and system-specific fonts to avoid exhibiting uncommon fingerprints. We also crawled the Fedora and Ubuntu Linux repositories to download all the plugins for both the i686 and x86_64 architectures. The biggest challenge is to account for the wealth of browser, operating system and CPU architecture combinations necessary for a plugin to function correctly in each possible DPC configuration, and to handle dependencies.

4.3.2.2 Modeling the well-formedness constraints in the diversity reservoir

We use feature modeling to ensure DPC configurations. The attributed feature model defines the valid configurations, i.e. configurations that can fundamentally run. We provide an extract of the feature model in Figure 4.4. A DPC configuration is composed of 4 mandatory features: an OS, a browser, plugins and fonts. It must be noted that some features in this model have attributes that specify the feature's characteristics and have domains that define possible values. For example, the attribute *Type* in the feature *Plugin* specifies that a plugin is either of type Netscape Plugin API (NPAPI) or Pepper Plugin API (PPAPI). Relations in the feature model and cross-tree constraints (see bottom of Figure 4.4) specify the set of rules that must be fulfilled for a DPC configuration to be valid and "*runnable*". For example, the first constraint states that a browser with a 64 bit architecture implies a 64 bit OS.

[1] http://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html

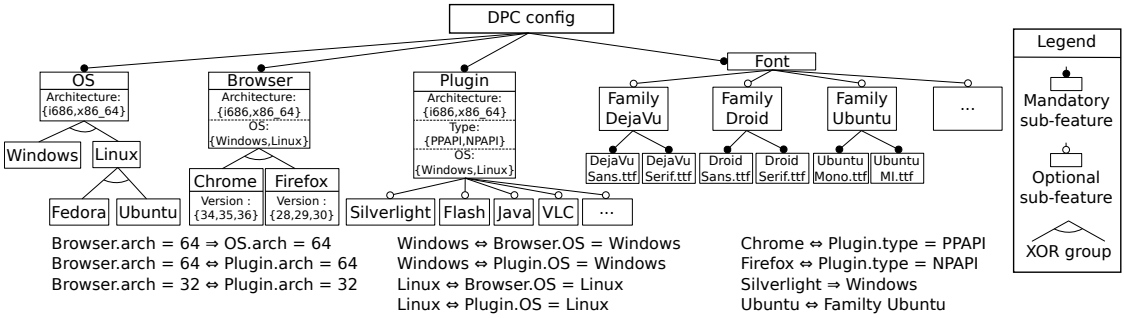


Figure 4.4: An extract of the feature model used for assembling valid DPC configurations.

4.3.3 Empirical validation of multi-level reconfiguration

We present a series of experiments to validate **multi-level platform reconfigurations are effective at breaking fingerprint linkability**. We show that the generated browsing platforms are diverse and we argue that consecutive fingerprints cannot be used to reidentify the same device, that is, they are not linkable.

Dataset. An **original platform** represents the user's browsing platform as it exists in the host operating system. The content of this platform can have a strong impact on Blink's ability at producing diverse platforms, since, in order to increase usability, we import the user's fonts and plugins when assembling a DPC configuration. In essence, the plugins and fonts of the user's platform are included as essential components in the assembled platforms. We test Blink's effectiveness using 25 original platforms. Each original platform has from 1 to 25 plugins, and from 20 to 520 fonts. The diversity reservoir for our experiments is: 4 Virtual machines (Fedora 20 32/64 bits, Ubuntu 14.04 32/64 bits); 2762 fonts; 39 browser plugins; 6 browsers, including 3 versions of Firefox: 28.0 (stable), 29.0 (beta), 30.0 (aurora), and 3 versions of Chrome: 34 (stable), 35 (beta), 36 (unstable).

Experimental protocol. For each of the 25 original platforms, we assemble 2 sequences of 100 platforms, providing a total of 5000 DPC configurations. For every assembled platform, we collect the exhibited fingerprint.

Results. To validate that the browsing platforms we assemble exhibit different fingerprints over time, we check that every consecutive platform exhibits fingerprints that are dissimilar. Thus, for each original platform, we collect the sequence of fingerprint dissimilarities by pairs of consecutive assembled platforms, using:

$$dissimilarity\_seq = (D(FP_i, FP_{i+1}))_{1 \leq i \leq 99}$$

For each of the 25 original platforms, we collect two sequences of dissimilarity values, which correspond to the 2 sequences of 100 assembled platforms. Figure 4.5 displays the distribution of dissimilarities for each original platform, in Leery mode. Figure 4.6 displays them in Coffee break mode. The X-axis in the figures correspond to the ID of the original platform, and the Y-axis represents the distribution of dissimilarities between pairs of consecutive platforms. The red line indicates the mean dissimilarity value among the 5000 collected fingerprints. Regarding the Leery strategy, Figure 4.5 shows that Blink is successful at assembling successive platforms that exhibit very dissimilar fingerprints, with results up to 0.77. Blink does sometimes assemble platforms that are very similar, with some pairs of successive platforms having dissimilarity values as low as 0.01. Since we have a relatively small pool of operating systems and browsers, it is likely that the OS or browser was the same from one configuration to the next. Yet, mean values are high thanks to our large pool of fonts and plugins. In Leery mode, dissimilarity varies between 0.01

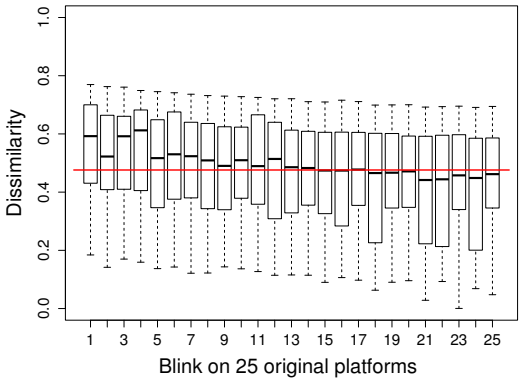


Figure 4.5: Dissimilarity between consecutive platforms (Leery mode)

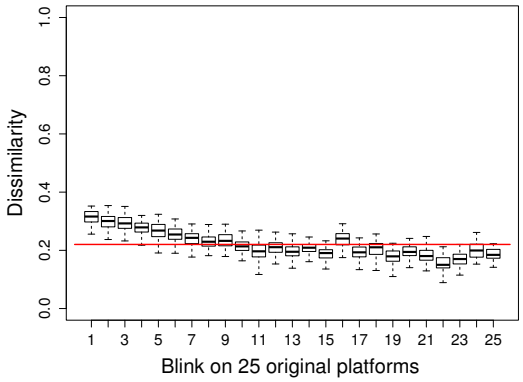


Figure 4.6: Dissimilarity between consecutive platforms (Coffee break mode)

and 0.77, while in Coffee break mode dissimilarity varies between 0.08 and 0.36. This shows the benefits of switching the operating system and browser. The mean dissimilarity values in Leery mode are more than twice as high as the ones in Coffee break mode.

We conclude from these observations that Blink generates configurations that are truly different from one another. It corresponds to our objective of breaking the stability of fingerprints over time, and shows its effectiveness in blocking the ability of fingerprinters to track users. Variations in Blink's effectiveness are due to the richness of the original platforms and on the size of the diversity reservoir.

4.3.4 Discussing Blink

Blink exploits automatic, multi-level reconfiguration and the natural diversity of software components in order to create a moving target defense against browser fingerprint tracking. This approach allows devices to exhibit a diversity of fingerprints without lying, and doesn't restrict the user's configuration or preferences like Tor, who attempts to create a unique fingerprint for all users and must limit the browser to do so. Blink breaks fingerprint *linkability*, and is resistant to cross-browser fingerprinting since it reconfigures low-level software like the OS.

It can be argued that using virtual machines to run browsers is costly. However, current computers can easily run multiple VMs simultaneously. Developers use them all the time. VMs allow Blink to target multiple platforms and architectures, and to reconfigure the OS, all important discriminators in fingerprints. Moreover, the use of VMs presents beneficial side-effects: it provides very strong sandboxing for all web activity. In our implementation, snapshots are used to return VMs to known safe-states, removing cookies and other browsing artifacts. In the end it's a tradeoff between performance, comfort and better privacy. The biggest impact to usability is the startup time of the VM. To improve startup times, we implemented a Linux only version using Docker. This led to much better startup times but the kernel and CPU architecture could not longer be reconfigured.

Finally, we didn't explore virtual machine, OS, browser, and hardware configurations, such as 2D and 3D acceleration. We believe that much more diversity could be easily included in an approach that randomizes individual configurations.

4.4 FP-TESTER: Automated Testing to Reduce Short-Term and Long-Term Fingerprintability

FP-TESTER is an approach to automatically test the effectiveness of browser fingerprinting countermeasure extensions. We implement a testing toolkit to help developers reduce fingerprint *uniqueness* and fingerprint *linkability*. More specifically, FP-TESTER is a testing toolkit that characterizes the fingerprintability of an extension at two levels:

1. *short-term fingerprintability—i.e.*, at time t, how different is the fingerprint from a browser with and without the extension installed;
2. *long-term fingerprintability—i.e.*, how does having the extension installed impact the ability to track a browser.

Although it will work on any browser extension, it is specifically important for countermeasures that aim to hinder tracking by changing or blocking attributes because they may easily introduce subtle side-effects that make browsers more identifiable [Starov 2017], sometimes to the point of rendering the extensions counterproductive (see 3.5). FP-TESTER reports on the side-effects introduced by the countermeasure, as well as how they impact tracking duration from a fingerprinter's point-of-view. To the best of our knowledge, FP-TESTER is the first tool to assist developers in fighting browser fingerprinting and reducing the exposure of end-users to such privacy leaks.

Figure 4.7 depicts an overview of FP-TESTER. We split the short and long term fingerprintability evaluations into two components, *short-term* and *long-term* fingerprintability, based on the idea, expressed in previous research [Laperdrix 2015, Nikiforakis 2015, Vastel 2018a], that fingerprint tracking requires fingerprints to be both *unique* and relatively stable over time to be *linkable*. The rest of this section is a summarized version of our paper FP-TESTER*: Automated Testing of Browser Fingerprint Resilience* published at EuroS&P in 2018 [Vastel 2018c] and describes our approach, implementation and experiments.

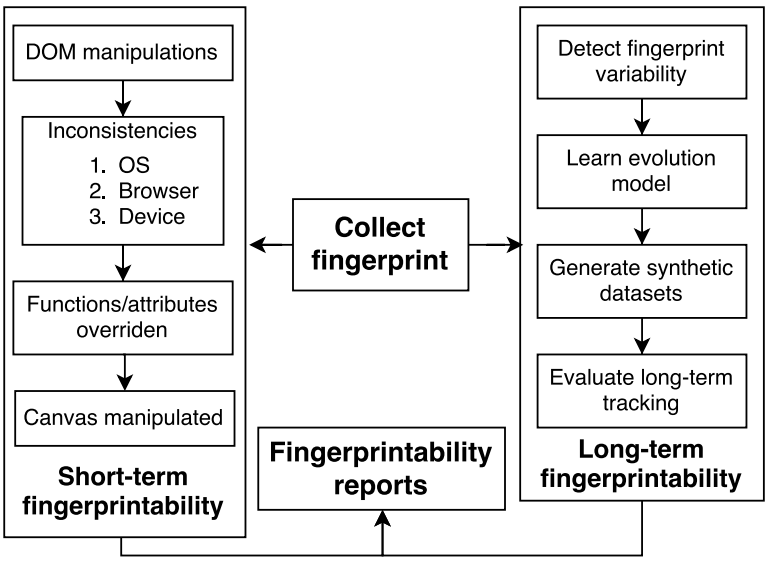


Figure 4.7: Overview of FP-TESTER toolkit

4.4.1 Short-Term Fingerprintability of Browser Extensions

Short-term fingerprintability reflects how, at a time t, the fingerprint generated by a browser with a countermeasure differs from a browser without it. We present 5 steps to evaluate short-term

fingerprintability.

DOM modifications We reuse the approach proposed by Starov *et al.* [Starov 2017] to monitor DOM changes made by extensions. For example, ad blockers may remove ads from the DOM.

Inconsistencies Fp-Tester also detects fingerprint inconsistencies. As shown in 3.5, countermeasures may introduce unexpected combinations of attributes, leading to *inconsistencies*. We propose a test suite to evaluate if the attributes of a fingerprint are globally consistent. These tests range from verifying that the user agent in the HTTP headers and the `navigator` object are coherent, to more subtle tests to verify that the rendering of an emoji in a canvas is consistent with the OS.

HTTP headers modification Fp-Tester scans HTTP headers to extract non-standard headers added by extensions.

Overridden functions/attributes Many extensions rely on overriding functions or getters involved in the fingerprinting process. We test if native JavaScript functions are overridden by looking at their `toString` representations.

Canvas fingerprinting Canvas fingerprinting has high entropy (see 3.3) and is central to countermeasures, such as Canvas Defender. Fp-Tester analyses the pixels to detect if they have been altered.

4.4.2 Long-Term Fingerprintability of Browser Extensions

Uniqueness is a requirement for tracking, and an extension may increase uniqueness, but this may be insufficient for tracking if, for example, the extension frequently changes the fingerprint. For example, a canvas randomizing extension may create unique images, but continual randomization may still make tracking impossible. However, if the extension has a very small user-base, none of this might matter, simply detecting it may increase trackability.

Detecting variability Fp-Tester detects the variability introduced by an extension. To detect the changes an extension introduces, we collect fingerprints from the browser with and without the extension activated. Multiple fingerprints are needed if the extension changes or randomizes the fingerprints between executions. This allows Fp-Tester to learn how the extension alters the browser fingerprint. Because extensions may be configurable, different settings may influence the fingerprint. Developers can specify a profile parameter, which allows Fp-Tester to test and learn the variability model for each configuration.

Evaluating long-term tracking Once Fp-Tester learns the extension's variability models, it evaluates how the extension exposes browsers to long-term tracking. Using genuine fingerprints obtained from the AmIUnique database [AmIUnique 2021], Fp-Tester simulates the presence of the countermeasure in browser fingerprints and quantifies the effect the countermeasure would have on tracking algorithms. Simulating the extension's effects on genuine fingerprints allows testing a wide range of configurations, without the overhead and difficulty of testing thousands of individual devices and browsers. To do so, Fp-Tester creates a population P composed of N genuine fingerprints, without inconsistencies, chosen randomly. This is necessary because most browsers do not use countermeasures, thus we attempt to recreate a realistic population to compare with. We randomly split P into 2 subpopulations, P_e and P_n, corresponding to the population of browsers with and without the extension, with a ratio f_e and f_n of N, respectively. Fp-Tester applies changes to all fingerprints in P_e so that they appear to originate from browsers with the countermeasure (detailed in Section 4.4.2). Using all fingerprints in P, Fp-Tester

executes the 2 linking algorithms from FP-STALKER [Vastel 2018b], as well as a new algorithm that leverages inconsistencies revealed by a browser to target it more efficiently. This allows comparing the vulnerability to tracking of browsers from P_n (without the extension), to browsers from P_e (with the extension).

Simulating Extension Changes on Fingerprints FP-TESTER simulates the effects of a countermeasure extension on genuine fingerprints. This process is complex and we do not claim to perfectly simulate the countermeasures since it may be difficult to recreate all their effects. However, even with imperfections, the effects are similar from the point-of-view of the tracking algorithms. As proposed in [Vastel 2018b], in order to decide if two fingerprints originate from the same browser, the algorithms generate a feature vector that is mostly pairwise comparisons of attributes between each fingerprint. For example, in the case of a canvas countermeasure, FP-TESTER may have learned that `toDataURL` is always overridden, and that the canvas is randomly modified. Without generating a canvas that reproduces the precise effects of the countermeasure, FP-TESTER is capable of generating a fingerprint that simulates the use of the countermeasure and, when compared to another fingerprint, results in a feature vector that is equivalent to one obtained if we had installed the countermeasure in the browser. Since, the feature vector used by the algorithms tests for the equality of the canvas attributes between two fingerprints, we assign a random value to the canvas, as it only needs to be different from the previous values, and we keep the fact that `toDataURL` is overridden. For attributes that are split into sub-attributes, with particular semantics, special attention is needed to reproduce the countermeasure's effects. It is not sufficient to simply detect that the attribute changed since the feature vector relies on more fine-grained details. FP-TESTER must learn how to apply changes to these attributes. For example, the user agent reflects the browser, its version, and the operating system. In this case, a change may be a random substitution from a list of known user agents, or a more subtle change in the browser version, or something in between. For such attributes, FP-TESTER discriminates different types of changes to different sub-attributes and attempts to reproduce these changes in the simulated fingerprints so that the comparisons remain similar. Furthermore, the choice to apply a change or not at a given time depends on the frequency determined during the learning phase of the variability models. Some extensions change attributes very frequently, while others only once.

4.4.3 Fingerprintability Reports and Components

We deliver actionable feedback to developers. Regarding short-term fingerprintability, we:

1. provide the modifications applied to the DOM,
2. provide a list of inconsistencies,
3. indicate which functions are overridden and how to hide them, and
4. we indicate if FP-TESTER detected a canvas manipulation.

Regarding long-term fingerprintability, we provide projections of how tracking algorithms would perform on the normal population versus the population that emulates browsers with the countermeasure. We also test if differences exist in terms of tracking duration depending on the extension profile.

Fingerprinter component FP-TESTER provides a fingerprinting script that allows to easily include new attributes.

Short-term components return `ShortTermResult` objects. This standardizes the sub-component test results and contains name, description, category (*e.g.*, overridden, inconsistency), as well as whether the test succeeded and a message to explain what it implies. The outputs are automatically integrated into the short-term fingerprintability report. One such sub-component is for emoji verification and uses supervised machine learning to verify that a rendered emoji is

consistent with the OS. We do not directly map the raw values to a specific OS as, for different devices on the same OS the emoji image differs. We trained our classifier on part of the canvas images from the AmIUnique dataset. Since the dataset contains years of data, it is suitable to train a robust classifier. The classifier is based on a *convolutional neural network* given as an example to classify images from the CIFAR10 dataset, which we retrained on our dataset. Concerning inconsistencies, since they play a central role in our approach, we provide a mechanism to add new consistency rules. We define a consistency rule as tuple of 3 attributes `(att1, att2, link)` where `att1` and `att2` are the two attributes which have a consistency link, and `link` is a function that takes the two attributes as parameters, and returns `true` if they are consistent. As an example, the user agent can be obtained from the `navigator` object and the HTTP headers, the tuple is defined as `(uaHttp, uaNavigator, eq)` where `eq` is the function that determines if the attributes are equal.

Long-term components are responsible for tasks such as the detection of variability introduced by a countermeasure. Contrary to the short-term component, we do not support adding new sub-components to it. Nevertheless, it is still possible to extend it by adding new tracking algorithms.

Report component is responsible for the generation of developer reports. Since values returned by short and long-term components are standardized, they can automatically be integrated into the report. This component can be extended to add new visualizations.

4.4.4 Preliminary Results: Random Agent Spoofer

We focus on the Random Agent Spoofer (RAS) for Firefox and detail the actionable feedback from Fp-Tester. RAS is an extension that aims at protecting against browser fingerprinting. RAS changes the values of attributes, such as the `user agent`, `platform`, the list of `languages`, and `screen resolution`. To avoid inconsistencies, RAS relies on a system of profiles—*i.e.*, combinations of attributes extracted from real browsers. It also allows disabling specific APIs, such as WebGL or WebRTC. We collected 72 fingerprints from browsers with RAS installed, each time we changed the profile.

Short-Term Fingerprintability Report After the collect phase, Fp-Tester looks for DOM manipulations, inconsistencies, overridden functions, and manipulated canvas pixels. Figure 4.8 presents the results of the analysis. Each node represents an attribute. A red edge between two nodes means that an inconsistency is detected between them. If a node's border is dashed, it means that Fp-Tester detects that the attribute has been overridden. In the case of RAS, Fp-Tester did not detect DOM or canvas pixel manipulations, nor non-standard HTTP headers, hence they are not in the image.

Inconsistencies in RAS Although attributes included in RAS's profiles are consistent among themselves, it is not the case of other attributes, such as Javascript errors. For each collected fingerprint, Fp-Tester extracts the browser, OS and device from the user agent to detect inconsistencies. This results in three types of inconsistencies:

- **OS inconsistencies:** Fp-Tester identified 6 tests that reveal OS inconsistencies (including the emoji test to verify the OS).
- **Browser inconsistencies:** RAS failed 3 tests to detect browser inconsistencies, including `eval.toString().length` (etsl, see 3.5.2).
- **Device inconsistencies:** RAS failed 2 tests, showing it spoofed the device. In particular, some fingerprints claimed to be mobile devices or tablets but did not support mobile-specific events, such as `touchSupport` or sensors like the accelerometer.

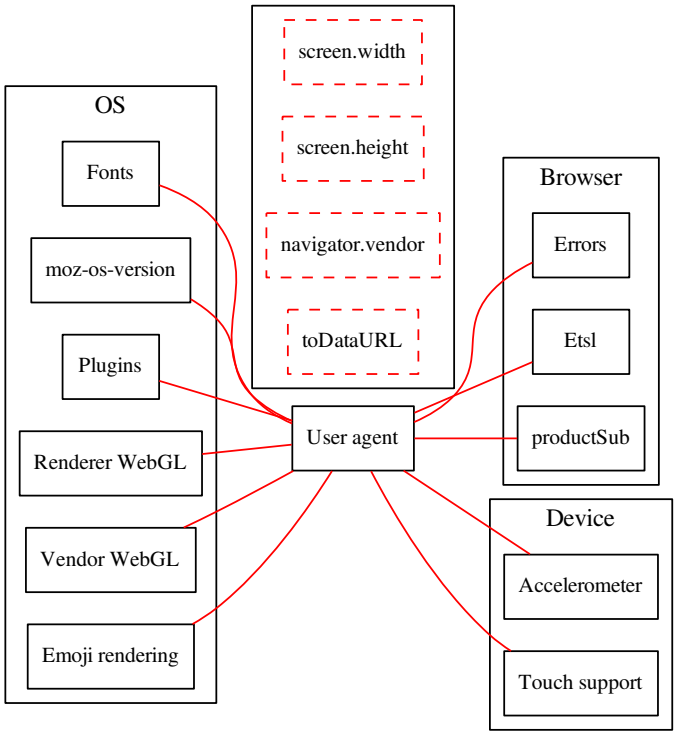


Figure 4.8: Short-term fingerprintability report provided by FP-TESTER

Overridden Functions by RAS FP-TESTER analyzed the `navigator` and `screen` prototypes and detected that the getters of `navigator.vendor` and `screen.width/height` were overridden. It also detected that `toDataURL` was also overridden.

Other Countermeasures We evaluated the short-term fingerprintability of four more countermeasures: CANVAS DEFENDER, CANVAS FINGERPRINT BLOCK, ULTIMATE USER AGENT SWITCHER (UUAS) and USER-AGENT SWITCHER (UAS). FP-TESTER detected inconsistencies introduced by all four extensions. Regarding canvas countermeasures, it detected overridden functions (`toDataURL`), as well as canvas pixels. UUAS is detected because it alters the user agent sent in the HTTP requests, but not the one in the `navigator` object. Although UAS changes both user agents, it does not modify other attributes, such as `navigator.platform`, which should be consistent with the OS.

4.4.5 Discussing FP-TESTER

FP-TESTERshows it's possible to detect countermeasures. Nevertheless, being detected with a countermeasure, or being unique, does not necessarily imply being tracked more easily, hence the distinction between short and long-term fingerprintability. We argue that the effect an extension may have on tracking depends on different factors, such as being able to identify the countermeasure, the number of users of the countermeasure, the ability to recover the real fingerprint values, and the volume of information leaked by the countermeasure. Thus, while it might not always be possible to provide an effective and undetectable countermeasure, it is important for countermeasure developers to evaluate if the anonymity gained from the countermeasure outweighs the information the countermeasure leaks. We believe FP-TESTER can help developers answer this question.

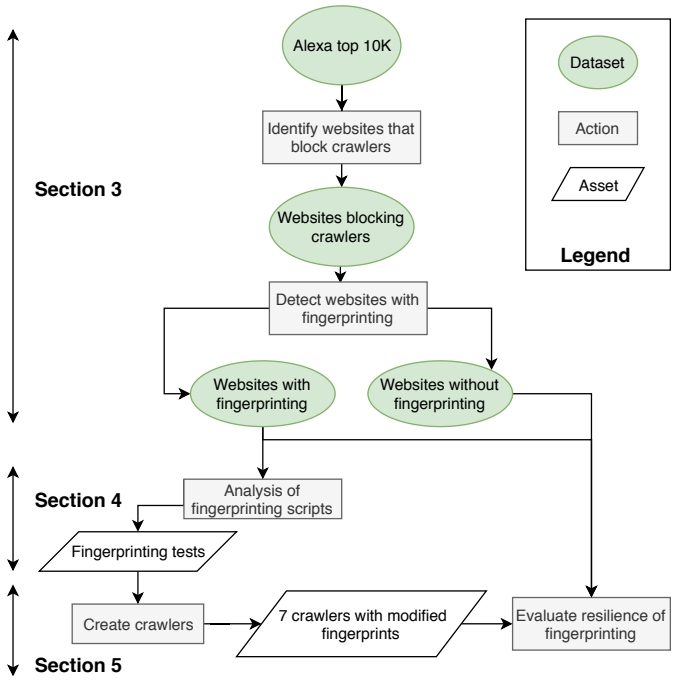


Figure 4.9: Overview of FP-Crawlers.

4.5 FP-Crawlers: exploiting browser fingerprint *consistency* to block crawlers

We're finally putting the *consistency* of browser fingerprints to use. FP-Crawlers studies how browser fingerprinting can be used to detect Web crawlers. Crawlers are a specific category of automated software bot that collect information from Web pages. For many different reasons, websites try to protect their information from crawlers. This information tends to be publicly accessible. Even when blocked behind registration or account creation, a Website might want to ensure that only humans are accessing the information. In FP-Crawlers, *uniqueness* and *linkability* are of less interest because we are not trying to track crawlers. However the *consistency* of fingerprinting is of keen interest because crawlers tend to spoof their attributes to bypass detection. As we saw in Section 3.5 this can often be detected. We show that fingerprinting is widely used to detect crawlers, it is much faster than comparable techniques, and it impacts human visitors less frequently. However, it faces serious limits against a motivated, competent attacker. An overview of FP-Crawlers is depicted in Figure 4.9. Our approach is pretty simple. We find crawler blocking fingerprinters, analyze the techniques they use, and evaluate the resilience of these against adversarial crawlers. The rest of this section is a summarized version of our paper FP-Crawlers*: Studying the Resilience of Browser Fingerprinting to Block Crawlers* [Vastel 2020], which won the Best Paper Award at MADWeb 2020.

4.5.1 Websites that block crawlers with fingerprinting

We begun our study by finding websites that block crawlers, and among those that do, identifying which ones use browser fingerprinting to do so and what specific techniques they use. We crawl Alexa's Top 10K with two crawlers, an *obvious to detect* crawler, `crawler1`, and a *slightly less obvious* crawler, `crawler2`, in parallel, on different IP addresses. We only crawl the homepage and up to 4 links to isolate fingerprinting from traffic analysis. Besides the IP address, the only different between the crawlers is the user agent. We use puppeteer [Google 2019], `crawler1`'s unchanged user agent is

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_2) AppleWebKit/537.36 (KHTML, like Gecko)
↪ HeadlessChrome/72.0.3582.0 Safari/537.36
```

You can see the user agent specifies `HeadlessChrome`. Because puppeteer is very popular, it's unlikely that websites that block crawlers don't check for this. Using such attributes that are specific to crawlers leads to no false positives. For `crawler2` we change the user agent to vanilla Chrome.

We label websites as "not blocked", "blocked", and "unknown". We manually inspect the "unknown" cases, in particular websites returing `HTTP 403`, resource forbidden, using a residential IP address and a desktop browser. From here, we are interested only in "blocked" websites. We revisit them with a new crawler, `crawler3`, with a vanilla chrome user agent and a `accept-language` header which is not sent by Headless Chrome by default. For each URL, the crawler records the attributes commonly accessed in the browser fingerprinting literature [Acar 2013, Gómez-Boix 2018, Englehardt 2016, Laperdrix 2016]. We override the properties of the `navigator` and the `screen` objects; functions related to canvas, audio, and WebGL fingerprinting; and access to attributes used by security fingerprinting scripts, such as `window._phantom` or `navigator.webdriver`, which are known to belong to crawlers. We explain the roles of these attributes in more detail in the next subsection. A complete list of the attributes and functions monitored is available in the full paper. Finally, we consider a website to use fingerprinting for crawler detection if:

1. At least one script called one or more functions related to canvas, WebGL, audio or WebRTC fingerprinting.
2. The script also tries to access one crawler-related attribute, such as `window._phantom` or `navigator.webdriver`.
3. The script also retrieves at least 12 fingerprinting attributes.

We adopt this definition as there is no clear agreement on how to characterize fingerprinting, in particular when used for crawler detection. For example, Acar *et al.* [Acar 2013] consider font enumeration as a good indicator. However, as we show that font enumeration is not the most discriminant feature for crawler detection. Our definition rather ensures that a script accesses a sufficient number of fingerprinting attributes, in particular attributes considered strong indicators of fingerprinting, such as canvas. As we study crawler detection, we add a constraint to check that the script accesses at least one crawler-related attribute, given that these are widely known and show intent to block crawlers [Eelmaa 2016].

4.5.2 Characterizing fingerprinting scripts

Among Alexa's Top 10K, we found 291 websites that block crawlers (2.91%). The median number of distinct fingerprinting attributes accessed is 12, while 10 % of the websites access more than 33. Concerning crawler-specific attributes (e.g., `navigator.webdriver`, `window._phantom`), 51.38 % of the websites do not access any, while 10% access 10. Based on our definition of browser fingerprinting, we found 93 that use fingerprinting for crawler detection, which represents 31.96% of the websites that block crawlers.

We group fingerprinting scripts by the combination of attributes they access. In total, we observe 20 distinct groups among websites blocking crawlers. While groups may contain essentially the same script from the same company on different sites, we also observe that some companies are present in different clusters because of multiple versions of their script. We focus on the scripts from 4 fingerprinting companies as they represent more than 90 % of the scripts among the websites that block crawlers. Since they have multiple versions of their script, we chose the script that accesses the greatest distinct number of fingerprinting attributes. We decided not to disclose the names of these companies since it does not contribute to the understanding of fingerprinting and our findings could be used by crawler developers to specifically target some websites. We deobfuscated the four scripts, and present the tests they perform in Table 4.3

We find that the scripts use a broad system of tests to identify crawlers. They detect if the crawler instrumentation framework injects **crawler specific attributes** into the HTML DOM

Table 4.3: Fingerprinting tests and the scripts that use them. ✓ indicates the attribute is collected and a verification test is run in the script. ∼ indicates the attribute is collected but no tests are run directly in the script. The absence of symbol indicates that the attribute is not collected by the script.

		Scripts			
	Name of the test	1	2	3	4
	Crawler-related attributes	✓	✓	✓	✓
Browser	productSub	∼	∼	∼	✓
	eval.toString()	✓			✓
	Error properties	✓			✓
	Browser-specific/prefixed APIs	✓	✓	✓	✓
	Basic features	✓		✓	✓
	Different feature behaviour				✓
	Codecs supported		✓		
	HTTP headers	∼	✓	∼	✓
OS	Touch screen support	∼	∼	∼	✓
	Oscpu and platform	∼	∼	∼	✓
	WebGL vendor and renderer	∼	∼		∼
	List of plugins	∼	∼	∼	✓
	List of fonts		∼	∼	
	Screen dimensions	✓	∼	∼	✓
	Overriden attributes/functions	✓	✓	✓	
Other	Events	∼			∼
	Crawler trap	✓			
	Red pill				✓
	Audio fingerprint	∼			
	Canvas fingerprint	∼	∼	∼	
	WebGL fingerprint	∼	∼		

or the JavaScript execution context, such as `_phantom`, `navigator.webdriver`, `__selenium_unwrapped` or `__webdriver_script_fn`. These do not incur false positives but are easy for crawlers to remove. Fingerprinters also verify numerous **inconsistencies**, such as error properties, `eval.toString`, overriden functions, OS inconsistencies, browser and version inconsistencies, and others that have been identified in FP-Scanner 3.5. For example, to test for browser-specific inconsistencies, they access `pushnotification` or `requestAnimationFrame`, audio and video codecs, and also verifying HTTP headers. In regards to device inconsistencies, they verify `navigator.maxTouchPoints` or `navigator.msMaxTouchPoints`, as well as the `navigator.oscpu` and `navigator.platform` attributes, and screen resolution inconsistencies.

Interestingly, we also tests to detect crawlers based on behavior (e.g., test for fake mouse movements), performance tests to identify emulators and virtual machines similar to the red pill presented by Ho et al. [Ho 2014], as well as a crawler trap using in invisible link with the `nofollow` property that appends a random UUID to the URL.

4.5.3 Effectiveness of browser fingerprinting to detect crawlers

Ground truth. In order to understand how effective fingerprinting is, we'd ideally have access to each of the websites blocking crawlers, and to the tests they perform on the backend. Since this is not possible, we run a family of crawlers on websites that have been identified as blocking crawlers. Thus, no matter how the crawler tries to alter its fingerprint, we can always assert that it is a crawler because it is under our control. Then, in order to measure the effectiveness of fingerprinting for crawler detection, we rely on the fact that the crawled websites have been identified as websites that block crawlers. We consider that, whenever they detect a crawler, they will block it. We use this blocking information as an oracle for the evaluation. Another solution

Table 4.4: List of crawlers and altered attributes.

Crawler	Attributes modified
Chrome headless based	
Crawler 1	User agent
Crawler 2	Crawler 1 + `webdriver`
Crawler 3	Crawler 2 + `accept-language`
Crawler 4	Crawler 3 + `window.chrome`
Crawler 5	Crawler 4 + `permissions`
Crawler 6	Crawler 5 + screen resolution + codecs + touch screen
Vanilla Chrome based	
Crawler 7	`webdriver`

to obtain the ground truth would have been to subscribe to the different bot detection services. Nevertheless, besides the significant cost, bot detection companies tend to verify the identity of their customers to ensure it is not used by competitors trying to reverse engineer their solution or by bot creators trying to obtain an oracle to maximize their ad-fraud incomes for example.

We crawl the websites with 7 different crawlers that incrementally modify their fingerprints to become increasingly more difficult to detect. The last crawler is based on a vanilla Chrome browser. We use this crawler to better understand why blocking occurs, and to assess that crawlers are blocked because of their fingerprint. Indeed, since this crawler is based on a vanilla Chrome, the only difference in its fingerprint is the `navigator.webdriver` attribute. Once this attribute is removed, it can no longer be detected through fingerprinting. Table 4.4 presents the crawlers and the attributes they modify. Each crawler builds on the previous one, the changes accumulate.

Crawling protocol. For each of the 7 crawlers, we run 5 crawls on the previously selected websites. Each crawl is run from a machine with a residential or university IP address that has not been used for crawling for at least 2 days to limit the influence of IP reputation. Studying how IP reputation influences detection is left as future work. A crawl consists of the following steps:

1. We randomly shuffle the order of the websites in the evaluation dataset. It enables to minimize and measure the side effects that can occur because of cross-domain detection;
2. For each website, the crawler visits the home page and then visits up to 10 randomly-selected pages from the same domain. As explained later in this section, we crawl only 10 links to ensure that we evaluate the effectiveness of browser fingerprinting detection and not the effectiveness of other state-of-the-art detection approaches;
3. Once a page is loaded, the crawler takes a screenshot and stores the HTML of the page for further analysis;
4. Between two consecutive crawled pages, the crawler waits for 15 seconds plus a random time between 1 and 5 seconds.

In total, we crawl 40 different websites, randomly selected from the list of websites blocking crawlers between December 2018 and January 2019. 22 of them use browser fingerprinting and 18 do not use fingerprinting. In total, we run 35 crawls—*i.e.*, 5 per crawler—each with a residential IP address that has not been used for at least two days for crawling.

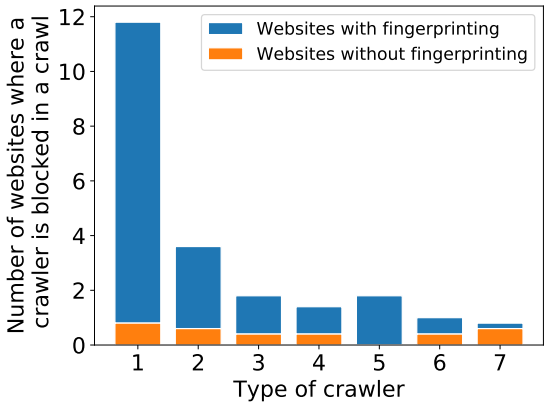


Figure 4.10: Crawl efficiency statistics.

Crawler blocking results. Figure 4.10 reports on the results of the crawls for the 7 crawlers. For each crawler, we present the average number of times per crawl it is blocked by websites that use fingerprinting and websites that do not use fingerprinting. We see that the more changes applied to the crawler's fingerprint, the less it is blocked. While Crawler 1 gets blocked 11.8 times on average, the detection falls to 1.0 time for Crawler 6 that applies more extensive modifications to its fingerprint. We also observe an important decrease in the number of times crawlers are blocked between crawlers 1 and 2. It goes from 11.8 for Crawler 1 to 3.6 for Crawler 2. The only difference being the removal of the `webdriver` attribute from the `navigator` object, which means that fingerprinters heavily rely on this attribute to detect crawlers.

Crawlers are blocked before crawling 3 pages of a website, on average. We also observe that, on average, websites using fingerprinting block more crawlers than websites without fingerprinting. For example, on average, 93.2 % (11.0) of websites blocking Crawler 1 use fingerprinting. The only exception is Crawler 7, where 75 % of the time it gets blocked, it is by a website not using fingerprinting. This is the expected result since Crawler 7 is based on a vanilla Chrome, which means that its fingerprint is not different from the one of a standard browser. The fact that Crawler 7 still gets blocked despite the fact it has a normal fingerprint raises the question of other detection factors used in addition to fingerprinting. Even though we take care to adapt the behavior of the crawlers to minimize the chance they get detected by other techniques, we cannot exclude that it occurs. Thus, we verify if crawlers are detected because of their fingerprint or because of other state-of-the-art detection techniques. First, we investigate if some of the crawlers have been blocked because of cross-domain detection. To do so, we manually label, for each fingerprinting script in the evaluation dataset, the company it belongs to. Whenever we cannot identify the company, we assign a random identifier. We identify 4 fingerprinters present on more than 2 websites in the evaluation dataset and that could use their presence on multiple domains to do cross-domain detection. We focus only on websites that blocked Crawlers 4, 5 and 6. Indeed, only one fingerprinting company succeeds to detect Crawlers 4, 5 and 6. Thus, we argue that Crawlers 1, 2 and 3 detected by websites using fingerprinting, are indeed detected because of their fingerprint. If their detection had relied on other techniques, then some of the Crawlers 4, 5, 6 and 7 would have also been blocked by these websites. Moreover, the analysis of the fingerprinting scripts we conducted shows that some of these fingerprinters have the information needed to detect Crawlers 1, 2 and 3, but not to detect more advanced crawlers using fingerprinting. Finally, we analyze in more detail the only fingerprinter that detected Crawlers 4, 5 and 6. At each crawl, the order of the websites is randomized. Thus, for each crawler and each crawl, we extract the rank of each of the websites that have a fingerprinting script from this company. Then, we test if the order in which websites from this fingerprinter are crawled impact the chance of a crawler to be detected. Nevertheless, we observe that crawlers get blocked on websites independently of their rank.

Non-stable blocking behavior. We also notice that websites that use the fingerprinting scripts provided by the only fingerprinter that blocked crawlers 4, 5 and 6 do not all behave the same way. Indeed, depending on the website, some of the advanced crawlers have never been blocked. It can occur for several reasons:

1. The websites have different versions of the scripts that collect different attributes;
2. On its website, the fingerprinter proposes different service plans. While some of them are oriented towards blocking crawlers, others only aim at detecting crawlers to improve the quality of the analytics data.

Even on the same website, the blocking behavior is not always stable over time. Indeed, some of the websites do not always block a given crawler. Moreover, some of the websites able to block advanced crawlers do not block crawlers easier to detect. For example, the only website that is able to block both crawlers 5 and 6, only blocked 13 times over the 35 crawls made by all the crawlers. It means that 37.1% of the time, this website did not block crawlers, even though it could have done so. In particular, this website never blocked Crawlers 1 and 2 even though they are easier to detect than Crawlers 5 and 6.

4.5.4 Discussing FP-Crawlers

We show that browser fingerprinting is relatively popular on Alexa's Top 10K. We also show that fingerprinting helps to detect more crawlers than non-fingerprinting techniques. For example, 93.2% (11 websites) of the websites that have detected crawler 1 use fingerprinting. Nevertheless, the important decrease in the average number of times crawlers are blocked between crawlers 1 and 2, from 11.8 websites to 3.6, indicates that websites mostly rely on simple features, such as the presence of the `webdriver` attribute, to block crawlers. We show that only 2.5% of the websites detect crawler 6 that applied heavier modifications to its fingerprint to escape detection, which shows one of the main flaws of fingerprinting for crawler detection: its lack of resilience against adversarial crawlers.

More generally, crawler technology, which used to be quite different than desktop browsers, has been catching up. Particularly with headless browsers, Chrome and Firefox, the gap between the desktop version and the headless version is getting smaller. Indeed, these changes require few lines of code (less than 300 lines in the case of Crawler 6) and can be done directly in JavaScript without the need to modify and compile a whole Chromium browser. This requires fingerprinting to be more invasive to detect the crawlers, but still can't stop adversarial crawlers from overcoming them. Ultimately, if a crawler wants to avoid being detected by fingerprinting methods, they should use a desktop browser. The reason they don't is arguably because it raises the cost and resources needed to run their crawls.

4.6 Summary

We provide a novel approach to avoid browser fingerprint tracking in *Blink*. Our insight comes from (i) seeing that most countermeasures can be detected because they spoof verifiable attributes, and (ii) seeing that there is so much diversity in Web clients that we can use it to our advantage and reconfigure a browsing platform very frequently, creating unique fingerprints each time that are difficult to link. Being unique for a short period of time, but very different after each change, allows us to *break fingerprint linkability*. We generate valid, consistent, correct browser platforms and all tools to increase their usability and customizability.

Fp-Tester is our approach to support developers in improving the resilience of their countermeasures against fingerprinting by identifying and *reducing uniqueness*. Our toolkit monitors DOM changes, errors, overridden native JavaScript functions, manipulated canvas pixels, and other inconsistencies in order to provide developer's with feedback on the *uniqueness* and *linkability* of their browsers and browser extensions.

In FP-Crawlers, we show browser fingerprinting is used to detect crawlers. Many scripts rely specifically on *exploiting fingerprint inconsistencies* in order to detect crawlers. We analyze the scripts from the main fingerprinters present in the Alexa Top 10K and show that they exploit features, errors and overridden native functions to detect crawlers. We implement 7 different crawlers and show that websites that fingerprint are better and faster at detecting crawlers compared to websites that use other state-of-the-art detection techniques. Nevertheless, while 29.5 % of the evaluated websites are able to detect our most naive crawler that applies only one change to its fingerprint, this rate decreases to 2.5% for the most advanced crawler that applies more extensive modifications to its fingerprint, showing that fingerprinting is not resilient to determined adversaries.

Doctoral student supervision associated to this chapter

Naif MEHANNA, 2020-2023 (100%, sole advisor)
Turning Ad Blockers Into Trackers.
Financed through the ANR JCJC FP-Locker project.

Antonin DUREY, 2018-2021 (50%, co-advised with Romain Rouvoy & Lionel Seinturier)
Leveraging Browser Fingerprinting to Fight Fraud on the Web.
Financed through a collaboration with the French Army Ministry.

Vikas MISHRA, 2018-2021 (50%, co-advised with Romain Rouvoy & L. Seinturier)
Contributions to tracking techniques : caches, IP addresses, fingerprinting.
Doctoral contract, Inria CORDIS.

Antoine VASTEL, 2016-2019 (66%, co-advised with Romain Rouvoy)
Tracking Versus Security: Investigating the Two Facets of Browser Fingerprinting.
Doctoral contract, Université de Lille
Defended October 2019. Currently working at Datadome in bot detection.

Benjamin DANGLOT, 2016-2019 (20%, co-advised with M. Monperrus & L. Seinturier)
Amplification automatique de tests unitaires pour DevOps.
Financed obtained by M. Monperrus through the H2020 STAMP project.
Defended November 2019, currently research Engineer.

Publications associated with this chapter

2021 Antonin Durey, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **FP-Redemption: Studying Browser Fingerprinting Adoption for the Sake of Web Security.** The 18th Conference on Detection of Intrusions and Malware & Vulnerability Assessment - DIMVA'21, Jul 2021, Lisboa, Portugal. https://hal.inria.fr/hal-03212726

2021 Vishra Mishra, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. **Déjà vu: Abusing Browser Cache Headers to Identify and Track Online Users.** PETS 2021 - The 21th International Symposium on Privacy Enhancing Technologies, July 2021, Virtual conference. https://hal.inria.fr/hal-03017222
Core Rank: B / Google Scholar (Computer Security): #15 Acceptance rate: 18%

2021 Antonin Durey, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **An iterative technique to identify browser fingerprinting scripts. 2021.** https://hal.inria.fr/hal-03212729

2020 Vikas Mishra, Pierre Laperdrix, Antoine Vastel, Walter Rudametkin, Romain Rouvoy, Martin Lopatka. **Don't count me out: On the relevance of IP addresses in the tracking ecosystem.** The Web Conference (TheWebConf'20, formerly WWW), Apr 2020, Taipei, Taiwan. https://hal.archives-ouvertes.fr/hal-02456195
Core Rank: A / Google Scholar (Databases & Information systems): #3 – Acceptance rate: 19%*

2020 David Zeber, Sarah Bird, Camila Oliveira, Walter Rudametkin, Ilana Segall, Fredrik Wollsén, Martin Lopatka. **The Representativeness of Automated Web Crawls as a Surrogate for Human Browsing.** The Web Conference (TheWebConf'20, formerly WWW), Apr 2020, Taipei, Taiwan. https://hal.archives-ouvertes.fr/hal-02456195
Core Rank: A / Google Scholar (Databases & Information systems): #3 – Acceptance rate: 19%*

2020 Antoine Vastel, Walter Rudametkin, Romain Rouvoy, Xavier Blanc. **FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers.** MADWeb'20 - NDSS Workshop on Measurements, Attacks, and Defenses for the Web, Feb 2020, San Diego, United States. https://hal.archives-ouvertes.fr/hal-02441653.
⧗ **Best Paper Award.**

2020 Sarah Bird and Vikas Mishra and Steven Englehardt and Rob Willoughby and David Zeber and Walter Rudametkin and Martin Lopatka, **Actions speak louder than words: Semi-supervised learning for browser fingerprinting detection**, 2020. https://arxiv.org/abs/2003.04463

2020 Benjamin Danglot, Martin Monperrus, Walter Rudametkin, et Benoit Baudry. **An approach and benchmark to detect behavioral changes of commits in continuous integration**. Empirical Software Engineering, March 5th, 2020. Pages 1573-7616. https://doi.org/10.1007/s10664-019-09794-7
Scimago Q1. Impact Factor 4.457. Google Scholar (Software Systems): #7

2018 Antoine Vastel, Walter Rudametkin, Romain Rouvoy. **FP-TESTER: Automated Testing of Browser Fingerprint Resilience.** 4th International Workshop on Privacy Engineering IWPE 2018, April 2018. https://hal.inria.fr/hal-01717158

2018 Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, Romain Rouvoy. **FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies.** 27th USENIX Security Symposium (USENIX Sec. 2018, Baltimore, USA). https://hal.inria.fr/hal-01820197
Core Rank: A/ Google Scholar (Computer Security): #4 – Acceptance rate: 19%.*

2016 Gustavo Sousa, Walter Rudametkin, Laurence Duchien. **Automated Setup of Multi-Cloud Environments for Microservices-Based Applications.** 9th IEEE International Conference on Cloud Computing (CLOUD), Jun 2016, San Francisco, USA. Pages 327-334. https://hal.inria.fr/hal-01312606
Core Rank: B – Acceptance rate: 15%.

2016 Inti Gonzalez-Herrera, Johann Bourcier, Walter Rudametkin, Olivier Barais, Francois Fouquet. **Squirrel: Architecture Driven Resource Management**. SAC - 31st Annual ACM Symposium on Applied Computing (SAC'16), Apr 2016, Pisa, Italy. Pages 1329-1336. https://hal.inria.fr/hal-01355000 *Core Rank: B*

2015 Pierre Laperdrix, Walter Rudametkin and Benoit Baudry. **Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification.** Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'15), Florence, Italy, May 2015. Pages 98-108. https://hal.inria.fr/hal-01121108 *Acceptance rate: 29%.*

2014 Inti Gonzalez-Herrera, Johann Bourcier, Erwan Daubert, Walter Rudametkin, Olivier Barais, François Fouquet, Jean-Marc Jezequel. **Scapegoat: an Adaptive monitoring framework for Component-based systems.** IEEE/IFIP Conf. on Software Architecture (WICSA 2014), Sydney, Australia. https://hal.inria.fr/hal-00983045 *Core rank A, Acceptance rate 21.1%.*

Research grants associated with this chapter

2019 **ANR JCJC FP-Locker : Hardening web authentication with browser fingerprinting** — Leader. The project's main objective is to develop an authentication system that uses browser fingerprinting to protect websites against fraud and bots. The project started in September 2020 with the recruitment of Naif Mehanna and includes members from the the CNRS and Mozilla. Ends 2024. Total funding: 168k€

2018 Research and collaboration contract with *Ministère des Armées* on the topic of **Leveraging Browser Fingerprinting to Fight Fraud on the Web**. Antonin Durey is funded under this contract. Total funding: 163k€.

2018 Inria ADT (*Action de Développement Technologique*) funding. The **FingerKit: a Cloud Platform to Study Browser Fingerprints at Large** project served to finance 24 engineer-months on a full rebuild of the AmIUnique platform with new functionality.

2018 Inria CORDI-S doctoral grant on **Collaborative Strategies to Protect from Browser Fingerprinting**. This grant finances Vikas Mishra's doctoral contract.

2016 Université de Lille doctoral grant on the subject of **Machine learning to optimize privacy against browser fingerprint tracking**. This grant financed Antoine Vastel's doctoral contract.

Chapter 5

Conclusions and Perspectives

To conclude, I provide a summary of the contributions reported in the manuscript and discuss some of the perspectives and the impact of my research in this area.

Contributions

In this manuscript I presented our contributions to browser fingerprinting. We have grouped said contributions into two chapters.

Chapter 3 focuses on the properties of fingerprinting that make it a risk to privacy, but also useful for security applications. In this chapter we have presented ou large-scale study, *Beauty and the Beast*, on fingerprint **uniqueness**, which is the capacity to uniquely identify a device by the characteristics that can be extracted through the browser. We show that it is not a perfect identification process but has a relatively high success rate. More surprisingly, we found that browser fingerprinting was almost as effective on mobile devices as it was on desktops and laptops, albeit for different reasons. The second property we study is **linkability**; the ability to link fingerprints from the same device over time. Because fingerprints evolve, it is not obvious that you can successfully link them. We show in Fp-Stalker that browser fingerprinting is imperfect. We are unable to link fingerprints from common, uncustomized devices for any period of interest for long-term tracking. In our dataset this represents about 20% of the devices. However, about 26% of the devices in our dataset are very trackable. Their fingerprints were linkable for their entire duration in the dataset. We hypothesize that people that customize their devices or use odd software and hardware configurations, for example, to protect themselves from tracking or commercial software that exfiltrates their data, are more likely to have linkable fingerprints. Over customizing and not using mainstream configurations makes you more identifiable, but doing nothing ensures you are trackable through stateful techniques such as cookies. It's a fine line to walk. The third property we study is fingerprint **consistency**. In Fp-Scanner we show that it's not easy to coherently spoof attributes in browser fingerprints because these modifications can often be detected. Being seen as a spoofer adds entropy to your fingerprint. Worse than that, in many cases, the true values can be revealed despite spoofing. We show that many of the countermeasures on the market, some widely popular, are quite deficient and sometimes even severely counterproductive to their intended privacy goals. However, interestingly enough, this same property makes verifications possible for security applications, a subject which we are actively exploring.

Chapter 4 presents our studies on practical defenses for, and uses of, browser fingerprinting. We present our very first project, called *Blink* [Laperdrix 2015], where we **break fingerprint linkability** by generating random browsing environments. A white-box, moving target defense to browser fingerprint tracking. By exploiting client-side diversity, we can make a usable browsing platform that exhibits genuine fingerprints, no spoofing required. In our preliminary work on automated browser extension testing, we present an extensible solution, Fp-Tester, to test short-term and long-term fingerprintability and provide feedback to developers to **reduce fingerprint uniqueness**. Finally, we take a look at Web crawlers and study how can be **exploit fingerprint consistency** to block them. We show it's fast, effective, but in practice not very resilient to a determined adversary.

Perspectives

In this section I quickly describe some of the perspectives to the work I presented in this manuscript. These perspectives build on much of the work we've done on browser fingerprinting, either by using it to secure websites or networks, or by discovering new fingerprinting attributes to increase unicity and linkability. I have already begun looking beyond browser fingerprinting towards other privacy risks, such as IP address tracking, geolocation, the TOR network, and even cookie-based tracking, in an attempt to better understand the totality of risks posed to users on the web. As of more recently, I've also begun to take an interest in the psychological and sociological impact of tracking, as well as some of the legal ramifications. I'm hoping that some of my recent collaborations will eventually result in a grant or two being finally accepted so we can explore these cross-discipline projects more thoroughly.

Multi-factor authentication. We are currently working on a large-scale deployment of browser fingerprinting to the centralized authentication server of a public institution. Browser fingerprinting will act as a second-level of verification for logins to the CAS server. The advantage of this use case, compared to, for example, crawler detection, is that users must provide their username and password for verification, and the fingerprint needs only be verified among the accepted devices for each user. We are working on manual and automated ways to assist users in adding devices to their list of "valid" devices, as well as ways of removing devices. We don't expect this to be perfect security solution, but we do hope to find it to be practical. It is particularly targeted for remote connections from untrusted networks. The fallback we imagine in case of a denied connection, maybe because the user switched devices, will be to switch to the VPN or connect using a trusted local network. We are currently working on the exact scenarios, which applications require better security, the user interface to manage fingerprints and devices, how we notify users, and all of the precise policies required for a large-scale deployment. We are also currently waiting for feedback from the CNIL. We hope that the project will succeed in identifying stolen credentials and fraudulent connection attempts, and also that the results will be satisfying enough to remove the yearly password change requirements currently established by the institute and detested by most of the users. This work is partially financed through the ANR JCJC FP-Locker project and the *Ministère des Armées* and is being implemented by Antonin Durey, a third-year PhD student I co-advise.

Finding new fingerprinting attributes. As is expected, a lot of research is done on finding new attributes that can be exploited. With the attempts to block or unify attributes in browsers, combined with the rapid addition of new APIs, the surface area for exploring new attacks is constantly changing. We have an ongoing collaboration with Ben-Gurion University of the Negev (BGU) in Israel and the University of Adelaide in Australia to identify hardware differences in graphics cards using hardware accelerated APIs. Not only the models but the individual graphics cards themselves. Our initial results show that there are performance differences that can be measured from otherwise identical hardware. This could serve to increase the identifiability of previously indistinguishable devices. We are also exploring the identification of ad-blockers and, more specifically, the filter lists they use. Our current results show that it is possibly to uniquely identify filter-lists and even the specific version of a filter-list. This would add entropy and, in cases where users have highly customized solutions, may potentially uniquely identify a device.

Combining tracking technologies. We have focused on isolating and studying browser fingerprinting by itself. However, we are very aware that most uses of browser fingerprinting would occur in conjunction with other techniques. We are particularly interested in the effectiveness of browser fingerprinting when combined with IPv4 and IPv6 addresses, as well as for cookie re-spawning for the few users that consistently erase their cookies. Last year we found that IPv4 address retention was a serious problem [Mishra 2020], it's likely that even naive fingerprinting approaches, combined with IP address geo-location, would have high success rates in

re-identifying users.

Crawler technology. We have been using crawling to understand the internet [Zeber 2020], study the pervasiveness of fingerprinting [Bird 2020, Durey 2021], or its effectiveness and resilience against crawlers [Vastel 2020]. We plan on continuing our work on crawlers, particularly exploring the construction of distributed, coordinated, stateful crawlers that can perform massive crawls and still avoid state-of-the-art bot detection measures. Furthermore, we plan on using crawlers to explore the use of free services, such as free proxies and VPNs, which might lure users for more nefarious purposes.

Automated testing. Starov et al. [Starov 2017] showed that browser extensions could be detected because of the way they interact with the DOM. In *Fp-Scanner* we explore a series of ways to identify fingerprinting countermeasures. And in *Fp-Tester*, we show that it's possible to assist developers in reducing the fingerprintability of their extensions. Our approach could be extended to all browser extensions, and even browsers. Furthermore, both browsers and many extensions are configurable, and many of these changes can impact the fingerprint exhibited by the browser. The advantages or automated testing are hard to understate and we believe a methodical, extensible and complete solution to automatically test the fingerprintability of browsers is warranted. Something that integrates into the development process, such as we have done previously, would be ideal [Danglot 2020].

Fingerprinting for lightweight security : Wireless access points & Internet of Things. In many wireless network deployments it can be difficult to monitor the network for new devices. Many of the solutions rely on MAC addresses that are easy to spoof. In the case of a network that verifies the MAC address before assigning an IP address, such as a laboratory or home network, fingerprinting could be used to verify it's the same device and not a spoofed address. In cases of campus-wide WiFi networks, fingerprinting could be used to detect stolen credentials. And in cases of WiFi access through secured portals, such as airport WiFi where the first 30 minutes are free, fingerprinting might be used to avoid MAC address spoofing used to avoid payment. We'd like to explore attacks where, unlike being limited by the protocols over the internet such as TCP/IP and HTTP, we have access to the data link or physical layers and combine the information with browser fingerprinting through, for example, a captive portal.

Smart buildings might use thousands of sensors and actuators that communicate over poorly secured networks. Many of these need to run for years, often a decade, before being changed out. The hardware limitations of these devices mean that their network stacks are limited and only partially implement specifications, such as IPv6. Furthermore, from one version of the firmware to the next, important and detectable changes can occur. We believe that fingerprinting could be used to identify devices, identify any spoofing on the network, and increase the overall security and understanding of the network.

Societal impact of omnipresent tracking. I have taken a personal interest in a phenomenon known as *social cooling*. The concept is simple. If you feel your being watched, you change your behavior. Among the things that change, are what you say and how you act. No longer having Through what is known as *surveillance capitalism*, many people are discovering that everything we do is being monitored, dissected and monetized. In a collaboration with Fabien Eloire, a sociologist, we have proposed a PhD subject to study user behavior and explore their actions as their knowledge of tracking changes[1]. We will perform large-scale automated studies of the web to identify and understand the tracking ecosystem and use this information to provide user-friendly, real-time feedback to users through novel tools that accurately present privacy-risks. We plan to perform controlled experiments to study changes in user behavior, in particular, informed v.

[1] https://www.pearl-phd-lille.eu/sites/www.pearl-phd-lille.eu/files/AAP%20PEARL%202/026%20-%20SOCIETAL_WEB_TRACKING.pdf

uninformed users, as well as their behavior with and without the assistance of our privacy-risk tools. Through rigorous controlled studies, surveys, interviews, questionnaires and large-scale crawls, we will understand user behavior in regards to privacy and hopefully find usable tools and methods to influence users towards safe, more privacy friendly actions. We believe the *AmIUnique* project is a excellent place to do these studies. Hopefully this will also shed light on the *privacy paradox*, which states that users that although users are concerned about privacy, their actions and behaviors do not mirror those concerns.

Dissemination, media coverage and impact

I'd like to conclude with some final remarks on the efforts my colleagues and I have put in to sharing our work and results with the general public. We have been running the AmIUnique (`https://amiunique.org`) project for almost 9 years. I have not mentioned it much throughout the manuscript other than to say it's the source of many of our datasets. This is in part a purposeful choice since I don't see it as a large scientific contribution in and of itself. It has, however, been a surprising success and an essential factor to most of our research in the area of browser fingerprinting. What originally started as a small project in Rennes, with a naive implementation running on a severely under provisioned virtual machine, has become essential for vulgarizing our research and the risks of browser fingerprinting, and has brought together more than 6 PhD students, who have all contributed to the project and benefited from the datasets and the visibility it provides. We have since brought the project to Lille, given it a much needed hardware upgrade, re-implemented the entire site to make it more scale. We currently have 2 to 3 thousand visitors a day and about 3.5 million fingerprints have been collected through the main site. Our browser extensions, less directly visible than the website, with close to 4 thousand users sending us their fingerprints every 4 hours, provide us with a high-quality, long-term dataset of millions of fingerprints. This has of course made papers such as *Beauty and the Beast*, FP-STALKER and FP-TESTER possible, as well as our work on IPv4 address retention [Mishra 2020], but more importantly, it's created a solid foundation on which we have built our research.

The AmIUnique project has also given us some perspective on the importance our research has and on the impact privacy issues have on the average citizen of the Web. We have had media coverage in the tech press, including articles, magazines and blogs[2,3,4,5], we are consistently mentioned in privacy articles,[6,7,8,9,10] we exchange frequently with the CNIL[11] and were awarded the CNIL Inria Privacy award in January 2019,[12] we have presented our results in general research[13] and computer science events,[14,15] open source communities like the Open Paris Summit

[2] https://framablog.org/2014/12/23/si-on-brouillait-les-pistes-avec-amiunique/
[3] https://yro.slashdot.org/story/14/12/14/1943218/how-identifiable-are-you-on-the-web
[4] https://connect.ed-diamond.com/MISC/MISC-081/Le-fingerprinting-une-nouvelle-technique-de-tracage
[5] https://www.clubic.com/pro/webmarketing/publicite-en-ligne/actualite-742853-fingerprinting-cookies.html
[6] https://www.nextinpact.com/news/105886-cookies-tiers-traceurs-fingerprint-et-compagnie-comment-ca-marche.htm
[7] https://arstechnica.com/information-technology/2017/02/now-sites-can-fingerprint-you-online-even-when-you-use-multiple-browsers/
[8] https://lifehacker.com/how-to-reclaim-your-digital-privacy-from-online-trackin-1820878546
[9] https://www.ionos.es/digitalguide/online-marketing/analisis-web/browser-fingerprinting-seguimiento-sin-cookies/
[10] https://www.computerworld.com/article/3339618/apple-ios/how-to-stay-as-private-as-possible-on-apples-ipad-and-iphone.html
[11] https://linc.cnil.fr/fr/b-baudry-p-laperdrix-et-w-rudametkin-il-est-essentiel-de-faire-de-la-recherche-sur-la-securite-et-la
[12] https://www.cnil.fr/fr/la-cnil-et-inria-decernent-le-prix-protection-de-la-vie-privee-2018
[13] https://journees-scientifiques2018.inria.fr
[14] https://www.min2rien.fr/journee-secu-le-jeudi-8-novembre-2018/
[15] https://www.elvigia.net/general/2016/11/10/inicia-congreso-internacional-ciencias-computacionales-cicomp-2016-254800.html

and fOSSa,[16] as well as cross-domain conferences,[17] and demos at the *Forum Internationale de la Cybercriminalité* in 2015 and in 2019.[18] This visibility has led to multiple research collaborations, including with Mozilla, the French Ministry of the Army, Ben-Gurion University of the Negev (BGU), CISPA, among others. It has also led to many student projects and even a recurring participation in the Inria's Hackatech Hackathon.[19] We plan to foster all of these collaborations, to continue to vulgarize our research, as well as maintain and improve the AmIUnique project for as long as browser fingerprinting is an issue that deserves attention.

As a final note, our work has taken on a more prescient role now that third-party cookies are to be deprecated[20], Google has warned about the risks that trackers begin to rely extensively on browser fingerprinting,[21] and while Google attempts to convince us they're the lesser evil and we should simply track ourselves and share the data.[22]

,

[16] https://www.lemonde.fr/sciences/article/2014/11/24/le-fossa-bazar-techno-participatif_4528568_1650684.html

[17] https://www.univ-rennes1.fr/evenements/18072019/colloque-le-profilage-en-ligne-construction-dune-approche-globale-lintersection-du-droit-de-linformatique-et-de-la-sociologie

[18] https://www.inria.fr/actualite/actualites-inria/fic-2019

[19] https://hackatechlille.inria.fr/technologies-inria/

[20] https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html

[21] https://www.blog.google/products/chrome/building-a-more-private-web/

[22] https://www.eff.org/deeplinks/2021/03/googles-floc-terrible-idea

Bibliography

[2Captcha 2018] 2Captcha. *Online CAPTCHA Solving and Image Recognition Service.*, 2018. (Cited on page 15.)

[Acar 2013] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gurses, Frank Piessens and Bart Preneel. *FPDetective: Dusting the web for fingerprinters.* In Proceedings of the ACM Conference on Computer and Communications Security, pages 1129–1140, 11 2013. (Cited on pages 12 and 67.)

[Acar 2014] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan and Claudia Diaz. *The Web Never Forgets: Persistent Tracking Mechanisms in the Wild.* Proceedings of the ACM Conference on Computer and Communications Security, pages 674–689, 11 2014. (Cited on pages 8, 9, 20, 21 and 38.)

[Acker 2017] Steven Van Acker and Andrei Sabelfeld. *Discovering Browser Extensions via Web Accessible Resources.* In CODASPY, 2017. (Cited on page 12.)

[AdBlock 2018] AdBlock. *AdBlock*, 2018. (Cited on page 13.)

[Adobe 2012] Adobe. *An Update on Flash Player and Android*, 2012. `https://blogs.adobe.com/flashplayer/2012/06/flash-player-and-android-update.html`. (Cited on page 23.)

[Alaca 2016] Furkan Alaca and P. C. van Oorschot. *Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods.* In Proceedings of the 32Nd Annual Conference on Computer Security Applications, ACSAC '16, pages 289–301, New York, NY, USA, 2016. ACM. (Cited on pages 12 and 38.)

[AmIUnique 2021] AmIUnique. *AmIUnique.org - Learn how identifiable you are on the Internet*, 2021. `https://amiunique.org/`. (Cited on pages 43 and 62.)

[Anderson 2017] Blake Anderson and David McGrew. *OS fingerprinting: New techniques and a study of information gain and obfuscation.* In 2017 IEEE Conference on Communications and Network Security (CNS), pages 1–9. IEEE, 2017. (Cited on page 14.)

[Augur 2018] Augur. *Augur Software*, 2018. (Cited on page 43.)

[Balla 2011] Andoena Balla, Athena Stassopoulou and Marios D Dikaiakos. *Real-time web crawler detection.* In Telecommunications (ICT), 2011 18th International Conference on, pages 428–432. IEEE, 2011. (Cited on page 15.)

[Bernard 2018] Benoit Bernard. *Web Scraping and Crawling Are Perfectly Legal, Right?*, 2018. (Cited on page 15.)

[Bird 2020] Sarah Bird, Vikas Mishra, Steven Englehardt, Rob Willoughby, David Zeber, Walter Rudametkin and Martin Lopatka. *Actions speak louder than words: Semi-supervised learning for browser fingerprinting detection*, 2020. (Cited on pages 2 and 77.)

[Bock 2017] Kevin Bock, Daven Patel, George Hughey and Dave Levin. *unCaptcha: a low-resource defeat of recaptcha's audio challenge.* In Proceedings of the 11th USENIX Conference on Offensive Technologies, pages 7–7. USENIX Association, 2017. (Cited on page 15.)

[Boda 2012] Károly Boda, Ádám Máté Földes, Gábor György Gulyás and Sándor Imre. *User Tracking on the Web via Cross-browser Fingerprinting.* In Proceedings of the 16th Nordic Conference on Information Security Technology for Applications, NordSec'11, pages 31–46, Berlin, Heidelberg, 2012. Springer-Verlag. (Cited on page 56.)

[Brave 2018] Brave. *Brave Software*, 2018. (Cited on page 44.)

[Breiman 2001] Leo Breiman. *Random Forests*. Machine Learning, vol. 45, no. 1, pages 5–32, 2001. (Cited on page 30.)

[Brotherson 2015] Lee Brotherson. *TLS fingerprinting*, 2015. (Cited on page 15.)

[Bursztein 2016] Elie Bursztein, Artem Malyshev, Tadek Pietraszek and Kurt Thomas. *Picasso: Lightweight Device Class Fingerprinting for Web Clients*. In Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '16, pages 93–102, New York, NY, USA, 2016. ACM. (Cited on page 12.)

[Canvas 2021] HTML Canvas. *HTML Canvas 2D Context*, 2021. http://www.w3.org/TR/2dcontext/. (Cited on page 21.)

[Cao 2017] Yinzhi Cao, Song Li and Erik Wijmans. *(Cross-)Browser Fingerprinting via OS and Hardware Level Features*. In NDSS, 01 2017. (Cited on page 9.)

[CAPTCHA 2018] Anti CAPTCHA. *Anti Captcha: captcha solving service. Bypass reCAPTCHA, FunCaptcha, image captcha.*, 2018. (Cited on page 15.)

[Chu 2013] Zi Chu, Steven Gianvecchio, Aaron Koehl, Haining Wang and Sushil Jajodia. *Blog or block: Detecting blog bots through behavioral biometrics*. Computer Networks, vol. 57, no. 3, pages 634–646, 2013. (Cited on page 15.)

[Commons 2016] Santa Clara Law Digital Commons. *Complaint for violation of the computer fraud and abuse act*, 2016. (Cited on page 15.)

[Cox 2006] Benjamin Cox, David Evans, Adrian Filipi, Jonathan Rowanhill, Wei Hu, Jack Davidson, John Knight, Anh Nguyen-Tuong and Jason Hiser. *N-variant systems: a secretless framework for security through diversity*. In Proc. of the Conf. on USENIX Security Symposium, USENIX-SS'06, 2006. (Cited on page 56.)

[Danglot 2020] Benjamin Danglot, Martin Monperrus, Walter Rudametkin and Benoit Baudry. *An approach and benchmark to detect behavioral changes of commits in continuous integration*. Empirical Software Engineering, vol. 25, no. 4, pages 2379–2415, July 2020. (Cited on page 77.)

[dillbyrne 2019] dillbyrne. *Random Agent Spoofer*, 2019. (Cited on page 13.)

[Docs 2018] MDN Web Docs. *MutationObserver API*, 2018. (Cited on page 46.)

[Durey 2021] Antonin Durey, Pierre Laperdrix, Walter Rudametkin and Romain Rouvoy. *An iterative technique to identify browser fingerprinting scripts*, 2021. (Cited on pages 2 and 77.)

[EasyList 2018] EasyList. *About EasyList*, 2018. (Cited on page 13.)

[EasyPrivacy 2018] EasyPrivacy. *EasyPrivacy*, 2018. (Cited on page 13.)

[Eckersley 2010] Peter Eckersley. *How Unique is Your Web Browser?* In Proceedings of the 10th International Conference on Privacy Enhancing Technologies, PETS'10, pages 1–18, Berlin, Heidelberg, 2010. Springer-Verlag. (Cited on pages 8, 10, 11, 12, 19, 35, 47, 54 and 56.)

[Eelmaa 2016] Erti-Chris Eelmaa. *Can a website detect when you are using selenium with chromedriver?*, 2016. (Cited on page 67.)

[Englehardt 2016] 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. (Cited on pages 9, 11, 13 and 67.)

[FaizKhademi 2015] Amin FaizKhademi, Mohammad Zulkernine and Komminist Weldemariam. *FPGuard: Detection and prevention of browser fingerprinting.* In IFIP Annual Conference on Data and Applications Security and Privacy, pages 293–308. Springer, 2015. (Cited on pages 14 and 43.)

[Fifield 2015] David Fifield and Serge Egelman. *Fingerprinting Web Users Through Font Metrics.* In International Conference on Financial Cryptography and Data Security, volume 8975, pages 107–124, 01 2015. (Cited on pages 9 and 42.)

[Firefox 2019] Firefox. *Firefox Public Data Report*, 2019. (Cited on page 13.)

[Foundation 2021a] Electronic Frontier Foundation. *Google's FLoC Is a Terrible Idea*, 2021. (Cited on page 8.)

[Foundation 2021b] Electronic Frontier Foundation. *Privacy Badger browser extension*, 2021. (Cited on page 13.)

[Frolov 2019] Sergey Frolov and Eric Wustrow. *The use of TLS in Censorship Circumvention.* In Network and Distributed System Security. The Internet Society, 2019. (Cited on page 14.)

[GmbH 2018] Eyeo GmbH. *Adblock Plus*, 2018. (Cited on page 13.)

[Gómez-Boix 2018] Alejandro Gómez-Boix, Pierre Laperdrix and Benoit Baudry. *Hiding in the Crowd: An Analysis of the Effectiveness of Browser Fingerprinting at Large Scale.* In Proceedings of the 2018 World Wide Web Conference, WWW '18, pages 309–318, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee. (Cited on pages 10, 14 and 67.)

[Google 2019] Google. *Puppeteer*, 2019. (Cited on page 66.)

[Hannak 2014] Aniko Hannak, Gary Soeller, David Lazer, Alan Mislove and Christo Wilson. *Measuring price discrimination and steering on e-commerce web sites.* In Proceedings of the 2014 conference on internet measurement conference, pages 305–318. ACM, 2014. (Cited on page 45.)

[Hill 2018] Raymond Hill. *uBlock Origin - An efficient blocker for Chromium and Firefox. Fast and lean.*, 2018. (Cited on page 13.)

[Hill 2019] Raymond Hill. *uMatrix: Point and click matrix to filter net requests according to source, destination and type*, 2019. (Cited on page 13.)

[Ho 2014] Grant Ho, Dan Boneh, Lucas Ballard and Niels Provos. *Tick Tock: Building Browser Red Pills from Timing Side Channels.* In WOOT, 2014. (Cited on pages 15 and 68.)

[Inc. 2018] Brave Software Inc. *Brave browser*, 2018. (Cited on pages 13 and 14.)

[Incapsula 2017] Imperva Incapsula. *Bot Traffic Report 2016.* `https://www.incapsula.com/blog/bot-traffic-report-2016.html`, January 2017. (Cited on page 15.)

[Jacob 2009] Gregoire Jacob and Christopher Kruegel. *PUB CRAWL : Protecting Users and Businesses from CRAWLers.* Protecting Users and Businesses from CRAWLers Gregoire, 2009. (Cited on page 15.)

[Jonker 2019] Hugo Jonker, Benjamin Krumnow and Gabry Vlot. *Fingerprint Surface-Based Detection of Web Bot Detectors.* In European Symposium on Research in Computer Security, pages 586–605. Springer, 2019. (Cited on page 12.)

[Jueckstock 2019] Jordan Jueckstock and Alexandros Kapravelos. *VisibleV8: In-browser Monitoring of JavaScript in the Wild.* In Proceedings of the Internet Measurement Conference, pages 393–405. ACM, 2019. (Cited on page 12.)

[kkapsner 2017] kkapsner. *CanvasBlocker: A Firefox Plugin to block the <canvas>-API*, July 2017. (Cited on page 14.)

[Laperdrix 2015] Pierre Laperdrix, Walter Rudametkin and Benoit Baudry. *Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification.* In Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '15, pages 98–108, Piscataway, NJ, USA, 2015. IEEE Press. (Cited on pages v, 1, 8, 43, 53, 61 and 75.)

[Laperdrix 2016] Pierre Laperdrix, Walter Rudametkin and Benoît Baudry. *Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints.* In 2016 IEEE Symposium on Security and Privacy (SP), pages 878–894, May 2016. (Cited on pages v, 1, 8, 11, 14, 18 and 67.)

[Laperdrix 2017] Pierre Laperdrix, Benoit Baudry and Vikas Mishra. *FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques.* In ESSoS 2017 - 9th International Symposium on Engineering Secure Software and Systems, page 17, Bonn, Germany, July 2017. (Cited on pages 14 and 44.)

[Laperdrix 2019] Pierre Laperdrix, Gildas Avoine, Benoit Baudry and Nick Nikiforakis. *Morellian Analysis for Browsers: Making Web Authentication Stronger with Canvas Fingerprinting.* In Detection of Intrusions and Malware, and Vulnerability Assessment - 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19-20, 2019, Proceedings, pages 43–66, June 2019. (Cited on page 12.)

[Loyola-González 2013] Octavio Loyola-González, Milton García-Borroto, Miguel Angel Medina-Pérez, José Fco Martínez-Trinidad, Jesús Ariel Carrasco-Ochoa and Guillermo De Ita. *An empirical study of oversampling and undersampling methods for lcmine an emerging pattern based classifier.* In Mexican Conference on Pattern Recognition, pages 264–273. Springer, 2013. (Cited on page 33.)

[Mayer 2009] Jonathan. R Mayer. *Any person... a pamphleteer.* Theses, Stanford University, 2009. (Cited on page 8.)

[Merzdovnik 2017] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker and Edgar Weippl. *Block Me if You Can: A Large-Scale Study of Tracker-Blocking Tools.* Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017, pages 319–333, 2017. (Cited on page 13.)

[Mishra 2020] Vikas Mishra, Pierre Laperdrix, Antoine Vastel, Walter Rudametkin, Romain Rouvoy and Martin Lopatka. *Don't count me out: On the relevance of IP addresses in the tracking ecosystem.* In Bruno Crispo and Nick Nikiforakis, editors, The Web Conference 2020, volume Security, Privacy, and Trust of *Proceedings of The Web Conference (WWW'20)*, Tapeï, Taiwan, April 2020. (Cited on pages v, 2, 76 and 78.)

[Mishra 2021] Vikas Mishra, Pierre Laperdrix, Walter Rudametkin and Romain Rouvoy. *Déjà vu: Abusing Browser Cache Headers to Identify and Track Online Users.* In PETS 2021 - The 21th International Symposium on Privacy Enhancing Technologies, Virtual, France, July 2021. (Cited on pages v and 2.)

[Mowery 2011] Keaton Mowery, Dillon Bogenreif, Scott Yilek and Hovav Shacham. *Fingerprinting Information in JavaScript Implementations.* In Helen Wang, editor, Proceedings of W2SP 2011. IEEE Computer Society, May 2011. (Cited on pages 9 and 10.)

[Mowery 2012] Keaton Mowery and Hovav Shacham. *Pixel Perfect: Fingerprinting Canvas in HTML5.* In Matt Fredrikson, editor, Proceedings of W2SP 2012. IEEE Computer Society, May 2012. (Cited on pages 9, 20, 21 and 41.)

[Mulazzani 2013] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, Edgar Weippl and FC Wien. *Fast and reliable browser identification with javascript engine fingerprinting.* In Web 2.0 Workshop on Security and Privacy (W2SP), volume 5, 2013. (Cited on pages 9 and 41.)

[Multilogin 2018] Multilogin. *Canvas Defender browser extension (canvas fingerprint blocker)*, 2018. (Cited on page 14.)

[networks 2018] Distil networks. *2018 Bad Bot Report.* `https://resources.distilnetworks.com/travel/2018-bad-bot-report`, January 2018. (Cited on page 15.)

[Nikiforakis 2013] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens and G. Vigna. *Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting.* In 2013 IEEE Symposium on Security and Privacy, pages 541–555, May 2013. (Cited on pages 9, 12, 13, 38, 41, 44 and 45.)

[Nikiforakis 2015] 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, WWW 2015, Florence, Italy, May 18-22, 2015, pages 820–830, 2015. (Cited on pages 13, 43 and 61.)

[NoScript 2021] NoScript. *NoScript browser extension*, 2021. (Cited on page 13.)

[Okhravi 2014] Hamed Okhravi, Thomas Hobson, David Bigelow and William Streilein. *Finding Focus in the Blur of Moving-Target Techniques.* Security Privacy, IEEE, vol. 12, no. 2, pages 16–26, Mar 2014. (Cited on page 54.)

[Olejnik 2016] Łukasz Olejnik, Gunes Acar, Claude Castelluccia and Claudia Diaz. *The Leaking Battery.* In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Alessandro Aldini, Fabio Martinelli and Neeraj Suri, editors, Data Privacy Management, and Security Assurance, pages 254–263, Cham, 2016. Springer International Publishing. (Cited on page 9.)

[Preuveneers 2015] Davy Preuveneers and Wouter Joosen. *SmartAuth: Dynamic Context Fingerprinting for Continuous User Authentication.* In Proceedings of the 30th Annual ACM Symposium on Applied Computing, SAC '15, pages 2185–2191, New York, NY, USA, 2015. ACM. (Cited on page 12.)

[Project 2020] The Tor Project. *Tor Browser*, 2020. (Cited on page 13.)

[Pujol 2015] Enric Pujol, Oliver Hohlfeld and Anja Feldmann. *Annoyed users: Ads and ad-block usage in the wild.* In Proceedings of the 2015 Internet Measurement Conference, pages 93–106. ACM, 2015. (Cited on page 13.)

[Quora 2018] Quora. *Is scraping and crawling to collect data illegal?*, 2018. (Cited on page 15.)

[Saito 2016] Takamichi Saito, Kazushi Takahashi, Koki Yasuda, Takayuki Ishikawa, Ko Takasu, Tomotaka Yamada, Naoki Takei and Rio Hosoi. *OS and Application Identification by Installed Fonts.* 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), pages 684–689, 2016. (Cited on page 40.)

[Sanchez-Rola 2018] Iskander Sanchez-Rola, Igor Santos and Davide Balzarotti. *Clock Around the Clock: Time-Based Device Fingerprinting.* In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, pages 1502–1514, New York, NY, USA, 2018. ACM. (Cited on page 10.)

[Schuh 2013] J. Schuh. *Saying Goodbye to Our Old Friend NPAPI*, September 2013. https://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html. (Cited on page 23.)

[Schwarz 2019] Michael Schwarz, Florian Lackner and Daniel Gruss. *JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits.* In NDSS, 2019. (Cited on page 12.)

[Sivakorn 2016] Suphannee Sivakorn, Jason Polakis and Angelos D Keromytis. *I'm not a human: Breaking the Google reCAPTCHA.* Black Hat, 2016. (Cited on page 15.)

[Sjösten 2017] Alexander Sjösten, Steven Van Acker and Andrei Sabelfeld. *Discovering Browser Extensions via Web Accessible Resources.* In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY '17, page 329–336, New York, NY, USA, 2017. Association for Computing Machinery. (Cited on page 10.)

[Sjösten 2018] Alexander Sjösten, Steven Van Acker, Pablo Picazo-Sanchez and Andrei Sabelfeld. *LATEX GLOVES: Protecting Browser Extensions from Probing and Revelation Attacks.* Power, page 57, 2018. (Cited on page 10.)

[Smart 2000] Matthew Smart, G Robert Malan and Farnam Jahanian. *Defeating TCP/IP Stack Fingerprinting.* In Usenix Security Symposium, 2000. (Cited on page 14.)

[Spooren 2015] Jan Spooren, Davy Preuveneers and Wouter Joosen. *Mobile Device Fingerprinting Considered Harmful for Risk-based Authentication.* In Proceedings of the Eighth European Workshop on System Security, EuroSec '15, pages 6:1–6:6, New York, NY, USA, 2015. ACM. (Cited on page 25.)

[Starov 2017] Oleksii Starov and Nick Nikiforakis. *Xhound: Quantifying the fingerprintability of browser extensions.* In 2017 IEEE Symposium on Security and Privacy (SP), pages 941–956. IEEE, 2017. (Cited on pages 10, 12, 61, 62 and 77.)

[Stassopoulou 2009] Athena Stassopoulou and Marios D Dikaiakos. *Web robot detection: A probabilistic reasoning approach.* Computer Networks, vol. 53, no. 3, pages 265–278, 2009. (Cited on page 15.)

[Stevanovic 2012] Dusan Stevanovic, Aijun An and Natalija Vlajic. *Feature evaluation for web crawler detection with data mining techniques.* Expert Systems with Applications, vol. 39, no. 10, pages 8707–8717, 2012. (Cited on page 15.)

[Takei 2015] N. Takei, T. Saito, K. Takasu and T. Yamada. *Web Browser Fingerprinting Using Only Cascading Style Sheets.* In 2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA), pages 57–63, Nov 2015. (Cited on page 40.)

[Torres 2015] Christof Ferreira Torres, Hugo Jonker and Sjouke Mauw. *FP-Block: Usable Web Privacy by Controlling Browser Fingerprinting.* In Günther Pernul, Peter Y A Ryan and Edgar Weippl, editors, Computer Security – ESORICS 2015, pages 3–19, Cham, 2015. Springer International Publishing. (Cited on pages 14 and 43.)

[Unger 2013] T. Unger, M. Mulazzani, D. Frühwirt, M. Huber, S. Schrittwieser and E. Weippl. *SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting.* In 2013 International Conference on Availability, Reliability and Security, pages 255–261, Sep. 2013. (Cited on page 12.)

[Unicode 2021] Unicode. *Emoji and Dingbats*, 2021. http://unicode.org/faq/emoji_dingbats.html. (Cited on page 21.)

[Van Goethem 2016] Tom Van Goethem, Wout Scheepers, Davy Preuveneers and Wouter Joosen. *Accelerometer-Based Device Fingerprinting for Multi-factor Mobile Authentication.* In Juan Caballero, Eric Bodden and Elias Athanasopoulos, editors, Engineering Secure Software and Systems, pages 106–121, Cham, 2016. Springer International Publishing. (Cited on page 12.)

[Vasilyev 2019] Valentin Vasilyev. *Modern and flexible browser fingerprinting library*, 2019. (Cited on page 43.)

[Vastel 2018a] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin and Romain Rouvoy. *FP-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies.* In 27th USENIX Security Symposium (USENIX Security 18), pages 135–150, Baltimore, MD, 2018. USENIX Association. (Cited on pages v, 1, 8, 19, 38 and 61.)

[Vastel 2018b] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin and Romain Rouvoy. *FP-STALKER: Tracking Browser Fingerprint Evolutions.* In Bryan Parno and Christopher Kruegel, editors, IEEE S&P 2018 - 39th IEEE Symposium on Security and Privacy, Proceedings of the 39th IEEE Symposium on Security and Privacy (S&P), pages 728–741, San Francisco, United States, May 2018. IEEE. (Cited on pages v, 1, 8, 18, 25 and 63.)

[Vastel 2018c] Antoine Vastel, Walter Rudametkin and Romain Rouvoy. *FP-TESTER: Automated Testing of Browser Fingerprint Resilience.* In 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pages 103–107. IEEE, 2018. (Cited on pages v, 1, 8 and 61.)

[Vastel 2018d] Antoine Vastel, Peter Snyder and Benjamin Livshits. *Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking.* arXiv preprint arXiv:1810.09160, 2018. (Cited on page 13.)

[Vastel 2020] Antoine Vastel, Walter Rudametkin, Romain Rouvoy and Xavier Blanc. *FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers.* In Oleksii Starov, Alexandros Kapravelos and Nick Nikiforakis, editors, MADWeb'20 - NDSS Workshop on Measurements, Attacks, and Defenses for the Web, San Diego, United States, February 2020. (Cited on pages v, 2, 8, 66 and 77.)

[Von Ahn 2003] Luis Von Ahn, Manuel Blum, Nicholas J Hopper and John Langford. *CAPTCHA: Using hard AI problems for security.* In International Conference on the Theory and Applications of Cryptographic Techniques, pages 294–311. Springer, 2003. (Cited on page 15.)

[Wang 2013] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng and Ben Y Zhao. *You Are How You Click: Clickstream Analysis for Sybil Detection.* In USENIX Security Symposium, volume 9, pages 1–008, 2013. (Cited on page 15.)

[Wikipedia 2013] Wikipedia. *Craigslist Inc. v. 3Taps Inc.*, 2013. (Cited on page 15.)

[Yen 2012] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu and Martin Abadi. *Host Fingerprinting and Tracking on the Web: Privacy and Security Implications.* Network and Distributed System Security Symposium, pages 1–16, 2012. (Cited on pages 7 and 9.)

[Yu 2016] Zhonghao Yu, Sam Macbeth, Konark Modi and Josep M. Pujol. *Tracking the Trackers.* In Proceedings of the 25th International Conference on World Wide Web, WWW '16, page 121–132, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee. (Cited on page 13.)

[Zalewski 2019] Michal Zalewski. *p0f v3*, 2019. (Cited on page 10.)

[Zeber 2020] David Zeber, Sarah Bird, Camila Oliveira, Walter Rudametkin, Ilana Segall, Fredrik Wollsén and Martin Lopatka. *The Representativeness of Automated Web Crawls as a Surrogate for Human Browsing*. In The Web Conference, Proceedings of The Web Conference (WWW'20), Taipei, Taiwan, April 2020. (Cited on pages 2 and 77.)

[Zhang 2013] Jing Zhang, Ari Chivukula, Michael Bailey, Manish Karir and Mingyan Liu. *Characterization of blacklists and tainted network traffic*. In International Conference on Passive and Active Network Measurement, pages 218–228. Springer, 2013. (Cited on page 15.)