Taming the Variability of Browser Fingerprints

Maxime Huyghe, Clément Quinton, Walter Rudametkin, Walter Rudametkin

Taming the Variability of Browser Fingerprints

Maxime Huyghe
maxime.huyghe@univ-lille.fr
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France

Clément Quinton
clement.quinton@univ-lille.fr
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France

Walter Rudametkin
walter.rudametkin@irisa.fr
Univ. Rennes, CNRS, Inria, IRISA
IUF
Rennes, France

ABSTRACT

Browser fingerprinting has become a prevalent technique for tracking and identifying users online, posing significant privacy risks. The increasing variability in web browser configurations, coupled with the continuous evolution of browser features, presents complex challenges in understanding and mitigating the impact of fingerprinting. In this paper, we introduce a novel approach that combines feature modeling techniques with tree-based representations to capture the intricate relationships and constraints within browser fingerprints. By translating 22,773 fingerprints into a feature model with 34,557 nodes, we enable a comprehensive analysis of their variability and uniqueness across 1,519 switches and 596 flags on 7 headless and headful browser versions. Our methodology facilitates various use cases, such as generating representative fingerprints for testing, detecting anomalies, and identifying discriminating attributes. We aim to provide developers and researchers with a powerful tool for studying browser fingerprints and developing effective strategies to enhance user privacy in the face of evolving tracking techniques.

CCS CONCEPTS

• **Security and privacy** → *Privacy protections*; • **Software and its engineering** → **Software product lines**.

KEYWORDS

Browser Fingerprinting, Configuration, Variability, Feature Model Synthesis

ACM Reference Format:
Maxime Huyghe, Clément Quinton, and Walter Rudametkin. 2024. Taming the Variability of Browser Fingerprints. In *Proceedings of 28th ACM International Systems and Software Product Line Conference (SPLC '24).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The browser is the primary interface through which we interact with online services and applications. However, the increasing complexity and variability of web browser configurations poses significant privacy and security challenges. Web browsers expose information about the user's device [6, 12, 22], operating system, fonts [10], extensions [28, 32], and various other characteristics. These can be used to create a unique identifier known as a browser fingerprint. While this information, exposed mainly through JavaScript APIs and attributes, allows websites to enhance and customize the user experience, it also raises concerns about their potential for tracking and profiling purposes. As web technologies evolve and new features are rapidly introduced, the variability of browser configurations increases, making it challenging to understand and manage the potential risks associated with the exposure of the browser's JavaScript attributes. Traditional approaches to studying browser configurations, such as storing them in JSON format [3, 23, 26, 27], do not adequately capture the complex relationships and constraints between different attributes nor their mappings to the browser's configuration.

We propose a novel approach that leverages feature modeling techniques to represent browser fingerprints in a structured and comprehensive manner. By translating each fingerprint into a tree structure and incorporating their constraints, we aim to gain a deeper understanding of the variability and uniqueness of browser fingerprints across different browser configurations. This representation allows for a more fine-grained analysis of the relationships between browser attributes and their impact on fingerprint distinctiveness. The main contributions of this paper are as follows:

- We introduce the idea of representing the browser fingerprint variability using the well-known technique of feature modeling for software variability [18].
- We propose algorithms for building individual feature trees from configurations and merging them into a unified tree, creating a comprehensive representation of the browser configuration landscape.
- We generated 22,773 browser fingerprints from Chromium. Our fingerprint dataset explores 1,519 switches and 596 flags from 7 versions of both headless and headful Chromium, which we use to build a feature model with 34,557 nodes.
- We provide insights into browser fingerprinting and discuss the implications of our approach for developing privacy-enhancing tools and assisting developers.

The remainder of this paper is structured as follows. Section 2 provides background information on web browser configurations and motivates the need for a structured representation. Section 3 explains how fingerprint variability is captured , and details the process of building and merging browser fingerprint feature trees. Section 4 describes various use cases of our approach and discusses the insights gained from our analyses. Section 5 presents related work in browser variability and feature model synthesis. Finally, Section 6 concludes the paper and outlines future research directions.

SPLC '24, September 2–6, 2024, Dommeldange, Luxembourg

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND AND MOTIVATION

Web browsers are used by billions of users worldwide and continuously change to adapt to new demands. Browser vendors integrate new functionalities by either developing them or by including third-party dependencies [20], resulting in increasingly complex source code. To adapt to demands, there is a growing number of alternative browsers, each offering their own sets of features. Some of these browsers have derivatives for mobile devices, ARM-based systems, and various operating systems. End users can tailor their browsing experience through the use of extensions, settings, and scripts. Each of these modifications contributes to the creation of unique configurations of the web browser, and every alteration has the potential to impact the browser's fingerprint.

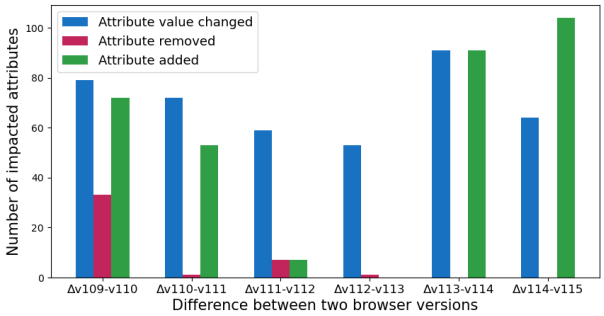
2.1 Browser Fingerprinting

Users employ web browsers for a wide range of activities, including browsing, streaming media, administrative tasks, social networking, and gaming. Each visited website can potentially retrieve a fingerprint [9] from the browser, enabling them to identify and track the device without relying on cookies or login information [8, 14]. A browser fingerprint is a unique set of attributes (keys) and their corresponding values, generated locally within the user's web browser and transmitted to the server, where it is stored and remains outside of the user's control. Certain attributes, such as the `Canvas` [17, 19] and the `User Agent` [13], are particularly popular and highly discriminating. The usage of browser fingerprinting has been increasing over time due to various factors, including for security, advertising, attacking, as well as for substituting or complementing the use of cookies due to increased user awareness about them and third-party cookie deprecation. As users continue to customize their browsing experiences, the uniqueness of their browser fingerprints increase. This technique has gained notoriety as a method for re-identifying users on the web, raising concerns about privacy and the potential for misuse.

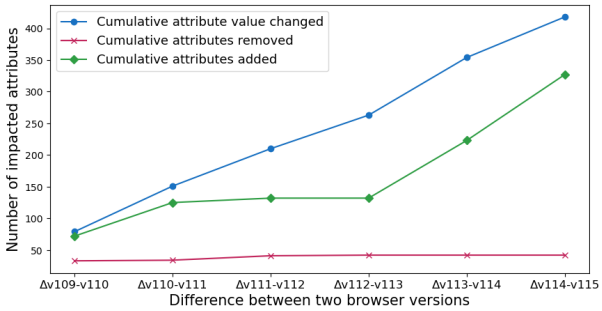
Browser fingerprints originate from the Browser Object Model (BOM).[1] Not to be confused with the Document Object Model (DOM), which focuses on the content of a web page, the BOM is a programming interface to interact with the browser and BOM provides access to objects, attributes, properties and methods related to the browser's window. For instance, screen size is accessible through the `window.screen.width` and `window.screen.height` attributes, and the language through `window.navigator.language`, which provide values such as `1920`, `1080`, and `en_US`, respectively. These values allow developers to adapt their applications to the device, but can also be used to identify the browser, or more specifically, the device. The risk increases as more attributes are considered.

2.2 Motivating Examples

The browser exposes a large set of attributes that can be exploited by attackers [8, 13]. Depending on the browser's version and configuration, 14,000 attributes can be collected [23]. In the state of the art, it is understood that browser fingerprints can be impacted by hardware components such as the CPU [22], audio setup [6], screen resolution [4], GPU [12], and others.They can also be affected by the operating system, fonts [10], and extensions [28, 32]. Furthermore, the variability in browser configurations (from *switches*[2] to *flags*, thousands of configuration parameters are available[3]) presents complex challenges in terms of user privacy and security. Fingerprints change as the browser evolves [31]. As depicted in Figure 1a, even with the same configuration, BOM attributes are added, removed, or their values changed between browser versions. Over 7 versions, as shown in Figure 1b, we observed many new attributes being added, few being removed.



(a) Attribute differences between consecutive browser versions.



(b) Cumulative attribute differences over consecutive browser versions.

Figure 1: Attribute evolution of Chromium's default configuration from versions 109 (baseline) to 115. Many attributes are added and values change, while few are removed.

Various use cases can leverage browser fingerprints. For example, for testing and debugging purposes, developers could randomly select and execute browser configurations that are representative of their user bases, or even attempt to approximate configurations that exhibit bugs. Also, anomalies and inconsistencies in fingerprints [30] may indicate potential security threats, *e.g.*, attributes that indicate bots or crawlers. Moreover, by analyzing the characteristics of browser fingerprints, it is possible to identify the most discriminating attributes, *i.e.*, rare attributes, attributes specific to some configurations, or attributes with high entropy. Identifying these would help developers reduce privacy risks. Finally, to manage the complexity of the BOM, the thousands of sources of variability, and the speed at which browser's evolve, developers require new approaches and automated support to analyze browser fingerprints.

[1] https://en.wikipedia.org/wiki/Browser_Object_Model

[2] https://peter.sh/feed/chromium-command-line-switches/

[3] https://chromium.googlesource.com/chromium/src/+/main/docs/configuration.md

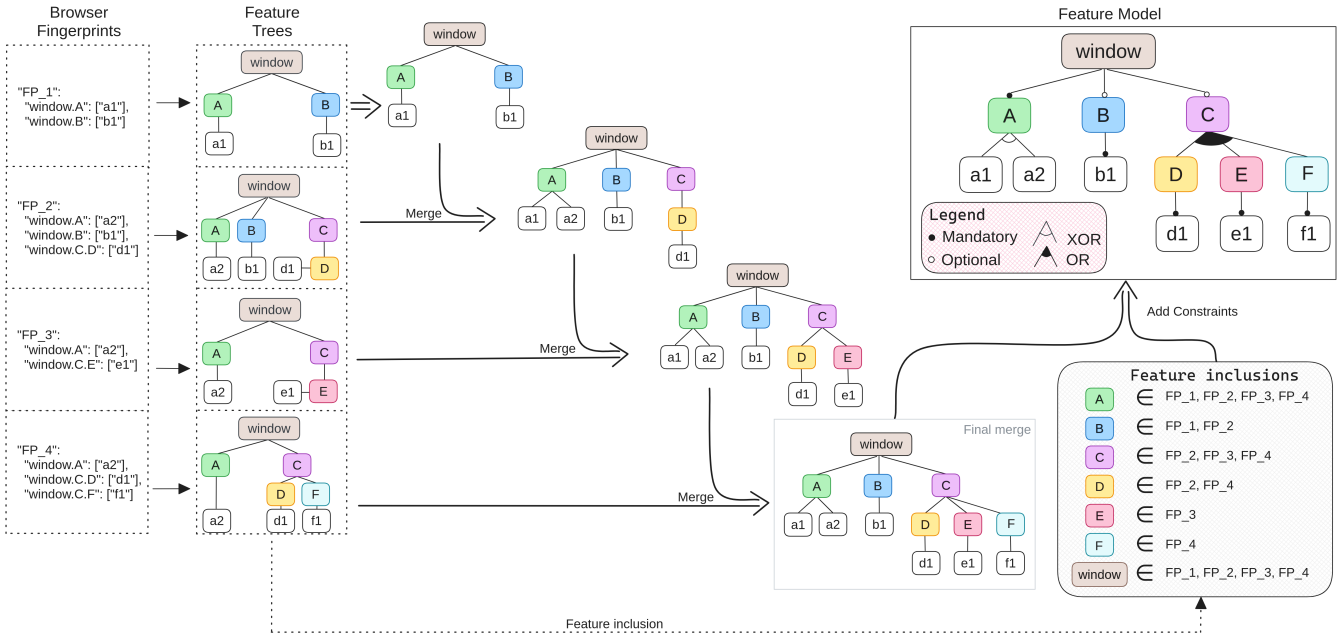


Figure 2: Feature Model Synthesis for Browser Fingerprints. We convert our dataset of fingerprints to feature trees. We record the list of configurations that contain a feature, which we call feature inclusions. Each feature tree is merged into a feature model. Mandatory, optional, XOR and OR constraints are calculated using parent-child relationships from the tree in conjunction with the list of feature inclusions.

3 CAPTURING FINGERPRINT VARIABILITY

To manage the complexity and changing nature of browser fingerprints, we propose an approach to synthesize a feature model that captures fingerprint variability[4]. This approach is threefold, as depicted by Figure 2. First, one feature tree is built from each fingerprint. Then, one by one, each feature tree is merged. The final feature tree is then refined with parent-child constraints that are calculated using the list of feature inclusions extracted from the fingerprints.

3.1 Building Feature Trees

A browser fingerprint is a set of attributes with values (*i.e.*, key-value pairs). We build a tree from a fingerprint using the following rules:

- Each attribute is translated into a node;
- Each value is translated into a leaf node;
- Each attribute contained in another attribute results in a parent-relationship for their respective nodes.

Figure 2 depicts the translation of 4 browser fingerprints into their respective feature trees. For instance, FP_2 consists of attributes `window.A` and `window.C.D` and two values, `a2` and `d1`. When applying the rules, a feature tree is created with `window` as the parent of both `A` and `C`, the latter being the parent of `D`, and two leaves, `a2` and `d1`. The same process is applied to each fingerprint, resulting in as many trees as fingerprints in the dataset. To build feature trees for the entire dataset (22,773 fingerprints), we parallelized the process using 24 threads, with each thread handling approximately 1,000 fingerprints. This approach reduced the building time to around 30 minutes on a desktop computer equipped with an AMD Ryzen 9 7900X and 32GB of RAM.

3.2 Merging Trees

To build the tree that represents all the fingerprints in the dataset, each fingerprint tree is merged according to Algorithm 1. We start with one tree from the dataset (line 3) and, for every remaining tree (lines 4 – 14), nodes are merged into the tree (lines 5 – 9). The addition of each new node is performed by looking for its parent (using its name, as each attribute name is unique) and adding that node as a child of the parent node (lines 7 – 9). If no parent exists, the node is the root (line 11). In practice `window` is always the root. Merging trees built from the 22,733 fingerprints results in a final tree with 34,557 nodes. The merging process can be applied to any new feature tree, enabling the continuous evolution of the final tree over time.

3.3 Refining the Feature Model

Once all feature trees are merged, we apply a refinement process to identify the parent-child constraints (*i.e.*, mandatory, optional, alternative or exclusive features). We rely on the concept of *feature inclusion*, which is similar to *feature degree* defined by Metzger et al. [16]. We define a feature inclusion $FI(f)$, for a given feature f, as the set of configurations that contain f. More precisely, we compute the feature inclusion for every feature by recording the set of fingerprints $FP = \{FP_1, ..., FP_n\}$ that contain the feature, in order

[4]https://doi.org/10.5281/zenodo.11519529

Algorithm 1 Merge trees

```
Input: trees                                  ▷ the set of feature trees
Output: mergedTree
mergedTree ← trees.get(0)
for tree ∈ trees\trees.get(0) do
    for node ∈ tree.nodes() do
        if ¬(mergedTree.contains(node)) then
            if node.hasParent() then
                parentNode ← node.getParent()
                mergedTree.get(parentNode).addChild(node)
            else
                mergedTree.setRoot(node)
            end if
        end if
    end for
end for
return mergedTree
```

to determine the constraints (see bottom-right of 2). To identify mandatory and optional constraints, we use the following:

- ***Mandatory* constraint**. $FI(f) = FI(f_{parent})$, that is, when a node is present in all configurations where its parent node is also present, then it is marked as mandatory in regards to its parent in the feature model. For instance, feature `A` is always present when `window` is present and is thus mandatory in regards to `window`.
- All other features are marked *optional* and, if their parent node has more than one optional child, will be checked for the stricter *XOR* and *OR* constraints.

For features that have more than one *optional* child feature, we refine the parent-child relationship. Let us consider $OPT(f_{parent}) = \{opt_{child_1}, ..., opt_{child_n}\}$ the set of optional children of a parent feature f_{parent}. Then, for all fingerprints in FP and for all OPT sets in a fingerprint:

- ***XOR* constraint**. If in every configuration where the parent node exists, there is always one optional child node for that parent node, that is, $FI(f_{parent}) = \bigcup_{i \in OPT(f_{parent})} FI(F_{child_i})$, and the child nodes never overlap among themselves, that is, $\forall x, y \in OPT(f_{parent}), x \neq y : FI(x) \cap FI(y) = \emptyset$ then the children of that parent node are mutually exclusive, otherwise known as XOR.
- ***OR* constraint**. Similarly, if in every configuration where a parent node exists, there is an optional child node for that parent node, that is, $FI(f_{parent}) = \bigcup_{i \in OPT(f_{parent})} FI(F_{child_i})$, but there exists at least one child node that overlaps with another, that is, $\forall x, y \in OPT(f_{parent}), x \neq y : \exists FI(x) \cap FI(y) \neq \emptyset$, then the children of that parent node are in an *OR* relationship.
- ***Optional* constraint**. All remaining features are given the optional constraint. This means that, in general, $FI(f_{parent}) \neq \bigcup_{i \in OPT(f_{parent})} FI(F_{child_i})$, that is, there is at least one child that does not respect a strict *XOR* relationship or a weaker *OR* relationship.

4 PRACTICAL APPLICATIONS

We have applied our feature model synthesis to a dataset of 22,773 browser fingerprints that we generated from 7 versions of the `Chromium` browser from 1,519 switches and 596 flags, on Linux, in two variants (graphical and headless), for a total of 14 browsers. To generate the fingerprints, we systematically explored distinct configuration parameters. The fingerprint dataset, with each file stored in a JSON format, uses 29 GB of space. Interestingly, the feature model version uses 3.2 GB (*i.e.*, 9 times smaller) when storing feature inclusions (*i.e.*, the list of configurations that initially contained the feature). If we remove the feature inclusions, which are only necessary for calculating constraints, the size decreases to 2 MB, which is approximately 14,500 times smaller. The resulting feature trees range from 12,311 to 16,018 features, with a constant depth of 8.

The feature model can be leveraged to address various scenarios, such as random sampling for testing or debugging purposes, detecting anomalies, tracking the evolution of browser fingerprints, or identifying discriminating attributes.

4.1 Sampling Fingerprints

As discussed in Section 2.2, web application developers may benefit from randomly selecting and executing different browser configurations for application testing or for validating security and privacy functions. However, fingerprint datasets are often too small or not representative of user devices. Researchers often rely on small datasets [4, 35] or collaborate with popular websites to include their scripts to collect real browser fingerprints [11]. Projects such as Electronic Frontier Foundation (EFF)[5] or Am I Unique[6] are also dedicated to fingerprint collections, but the datasets from these sites are: *(i) not shared* for privacy concerns, *(ii) not exhaustive* because it is challenging to collect a set of configurations large-enough to cover variations in hardware, user settings, operating systems, browser versions, and other factors, *(iii) not representative* of average users since visitors to such sites are more likely to be aware of browser fingerprinting techniques and take privacy measures to counter them. Additionally, privacy and user consent concerns can pose significant challenges in collecting and using browser fingerprint data. To tackle these issues, an alternative approach consists of generating fingerprints. However, fingerprint generation may result in invalid browser fingerprints [30]. By leveraging the synthesized feature model and applying existing sampling approaches [34], one can create a valid and relevant set of browser fingerprints to be used for testing purposes.

4.2 Fingerprint Evolution

Figure 1a illustrates the attribute changes in browser fingerprints from `Chromium` versions 109 to 115. In particular, version 114 looks like a turning point: attributes were removed regularly until this version, while added attributes increased from here. To observe fingerprint evolution, we first synthesize a feature model from the default browser version, *i.e.*, a browser configuration devoid of *switches* or modified *flags*, and we then compute the feature model differences between two of these versions [2]. Relying on

[5] https://coveryourtracks.eff.org/
[6] https://amiunique.org

the set of synthesized feature models, we can thus track removed features (*e.g.*, due to BOM deprecation), or measure the impact on user privacy when new features are added to the browser.

Our experiments show that the number of nodes increases over time. Between `Chromium` versions 109 and 115, *i.e.*, 7 releases over 6 months, we observed an increase of 1,333 new nodes, from 28,926 to 30,259 nodes. By leveraging the feature model of each browser version and applying random sampling techniques, browser developers can assess the privacy impact of newly implemented features. This approach allows them to explore a wide range of configurations, identify potential privacy risks, and make informed decisions about the design and implementation of new functionality.

4.3 Configuration Identification

To generate our fingerprint dataset, we exhaustively explored the following runtime configuration spaces[7]:

- **Switches**[8] are command-line arguments initialized before the browser's launch. We explored 1,519 switches.
- **Flags** are experimental features accessible via the browser's `chrome://flags` page. We explored 596 flags.

We observed that fingerprint feature nodes appear or disappear depending on the (de)activation of certain configuration parameters. These observations raised questions regarding whether these nodes could be identifiable and linked to the browser's configuration. To facilitate the identification of these configuration parameters, we computed differences between a feature tree resulting from the default configuration and feature trees generated from configurations where *switches* and *flags* were activated one by one. Computing these differences was performed relying on the method proposed in [2]. A *switch* or *flag* parameter was thus considered to impact the fingerprint when added or removed nodes resulted from that diff operator.

We stored a mapping for each *switch* and *flag* to their fingerprint. Some of these configuration parameters are highly identifiable because they enable or disable features in the web browser that impact the browser fingerprint and therefore our feature model. We also generated new browser fingerprints in more recent versions on different hardware and operating systems using *switches* that we knew were identifiable. Subsequently, we attempted to verify whether we could identify the *switch* used from the browser fingerprint. In the majority of cases, we successfully identified the *switch* from the browser fingerprint. In instances where we did not correctly identify it, the set of nodes was too close or identical to another case. With this information, we should be able to identify browser configurations from their fingerprints and potentially provide tips on how to make the configuration more privacy-friendly, if desired. Another use case is for developers. If a user reports a bug and generates a report, we can easily add the browser fingerprint to the report. This allows the developer to reproduce the user's configuration, gaining a better understanding of the bug, and correcting it more efficiently.

[7] https://chromium.googlesource.com/chromium/src/+/main/docs/configuration.md
[8] https://peter.sh/feed/chromium-command-line-switches/

5 RELATED WORK

On Browser Variability. In the context of browser configurations, Swanson et al. [29] proposed the `REFRACT` framework to avoid failures. Their approach involves adding guards to monitor each configuration and creating valid configurations when failures occur. While their work focuses on configuration testing, it highlights the importance of managing browser variability. Beyond the software engineering domain, the application of feature modeling has been explored in other fields. Cashman et al. [5] proposed importing the software product line approach to the `Biobrick` repository in the biological domain, facilitating the planning of DNA element reuse. This demonstrates the potential of applying feature modeling techniques to diverse problem domains that involve managing variability, similar to the challenges faced in browser fingerprinting.

Feature Model Synthesis. Ryssel et al. [21] employed concept analysis to generate feature models that incorporate or-groups and xor-groups. Their work automatically extracts meaningful feature relationships from existing configurations. Similarly, She et al. [25] tackled the problem of automatic model synthesis from propositional constraints, proving its NP-hardness and proposing enhanced formulas in both conjunctive normal form (CNF) and disjunctive normal form (DNF). The application of evolutionary algorithms to reverse engineer feature models has been explored by Lopez-Herrejon et al. [15]. Their study provides valuable insights into the potential for reconstructing feature models from existing systems. Acher et al. [1] proposed a systematic procedure for extracting feature models from product descriptions, enabling a more structured and efficient modeling process. In the domain of requirements engineering, Weston et al. [33] introduced a comprehensive framework to support software product line engineers in constructing feature models based on natural language requirements. Their work aims to bridge the gap between textual specifications and formal feature representations. Additionally, She et al. [24] presented a novel procedure that simplifies the reverse engineering process of feature models. They introduced a ranking heuristic to identify parent candidates and automated procedures for detecting mandatory features, feature groups, and implies/excludes relationships. Czarnecki et al. [7] proposed an algorithm based on binary decision diagrams for synthesizing feature models from logical formulae, thus leveraging formal methods to automate the feature modeling building process.

6 CONCLUSION

In this work, we introduced a novel approach for capturing the variability and complexity of browser fingerprints through the application of feature modeling and tree-based representations. By employing this novel feature modeling approach, we have developed a structured methodology for comprehensively capturing the nuances of browser fingerprints. This not only enhances our understanding of fingerprint variability but also paves the way for developing more effective privacy-preserving solutions and anomaly detection techniques. We successfully created a feature model with 34,557 nodes from 22,773 browser fingerprints generated from the exhaustive exploration of 1,519 switches and 596 flags in both headless and GUI variants of 7 versions of Chrome.

REFERENCES

[1] Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. 2012. On extracting feature models from product descriptions. *Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems* (2012). https://doi.org/10.1145/2110147.2110153

[2] Mathieu Acher, Patrick Heymans, Philippe Collet, Clément Quinton, Philippe Lahire, and Philippe Merle. 2012. Feature Model Differences. In *Advanced Information Systems Engineering*, Jolita Ralyté, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 629–645. https://doi.org/10.1007/978-3-642-31095-9_41

[3] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. 2016. Disguised chromium browser: Robust browser, flash and canvas fingerprinting protection. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*. 37–46.

[4] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. 2012. User Tracking on the Web via Cross-Browser Fingerprinting. In *Information Security Technology for Applications*, Peeter Laud (Ed.). Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 31–46. https://doi.org/10.1007/978-3-642-29615-4_4

[5] Mikaela Cashman, Justin Firestone, Myra B. Cohen, Thammasak Thianniwet, and Wei Niu. 2019. DNA as Features: Organic Software Product Lines. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 108–118. https://doi.org/10.1145/3336294.3336298

[6] Shekhar Chalise, Hoang Dai Nguyen, and Phani Vadrevu. 2022. Your speaker or my snooper? measuring the effectiveness of web audio browser fingerprints. In *Proceedings of the 22nd ACM Internet Measurement Conference* (Nice, France) *(IMC '22)*. Association for Computing Machinery, New York, NY, USA, 349–357. https://doi.org/10.1145/3517745.3561435

[7] K. Czarnecki and Andrzej Wąsowski. 2007. Feature Diagrams and Logics: There and Back Again. *11th International Software Product Line Conference (SPLC 2007)* (2007), 23–34. https://doi.org/10.1109/SPLC.2007.19

[8] Peter Eckersley. 2010. How unique is your web browser?. In *Privacy Enhancing Technologies: 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings 10*. Springer, 1–18. https://doi.org/10.1007/978-3-642-14527-8_1

[9] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016-10-24) *(CCS '16)*. Association for Computing Machinery, 1388–1401. https://doi.org/10.1145/2976749.2978313

[10] David Fifield and Serge Egelman. 2015. Fingerprinting Web Users Through Font Metrics. In *Financial Cryptography and Data Security*, Rainer Böhme and Tatsuaki Okamoto (Eds.). Vol. 8975. Springer Berlin Heidelberg, 107–124. https://doi.org/10.1007/978-3-662-47854-7_7 Series Title: Lecture Notes in Computer Science.

[11] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *Proceedings of the 2018 world wide web conference*. 309–318. https://doi.org/10.1145/3178876.3186097

[12] Tomer Laor, Naif Mehanna, Antonin Durey, Vitaly Dyadyuk, Pierre Laperdrix, Clémentine Maurice, Yossi Oren, Romain Rouvoy, Walter Rudametkin, and Yuval Yarom. 2022. DRAWN APART : A Device Identification Technique based on Remote GPU Fingerprinting. In *Proceedings 2022 Network and Distributed System Security Symposium* (San Diego, CA, USA, 2022). Internet Society. https://doi.org/10.14722/ndss.2022.24093

[13] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser fingerprinting: A survey. *ACM Transactions on the Web (TWEB)* 14, 2 (2020), 1–33. https://doi.org/10.1145/3386040

[14] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In *37th IEEE Symposium on Security and Privacy (S&P 2016)*. San Jose, United States. https://inria.hal.science/hal-01285470

[15] Roberto Erick Lopez-Herrejon, José A. Galindo, David Benavides, Sergio Segura, and Alexander Egyed. 2012. Reverse Engineering Feature Models with Evolutionary Algorithms: An Exploratory Study. In *International Symposium on Search Based Software Engineering*. https://doi.org/10.1007/978-3-642-33119-0_13

[16] Andreas Metzger, Clément Quinton, Zoltán Ádám Mann, Luciano Baresi, and Klaus Pohl. 2020. Feature Model-Guided Online Reinforcement Learning for Self-Adaptive Services. In *Service-Oriented Computing*, Eleanna Kafeza, Boualem Benatallah, Fabio Martinelli, Hakim Hacid, Athman Bouguettaya, and Hamid Motahari (Eds.). Springer International Publishing, Cham, 269–286. https://doi.org/10.1007/978-3-030-65310-1_20

[17] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP* (2012). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3208feae829cba6bd319421fe1fea58962da8fd9

[18] Damir Nešić, Jacob Krüger, undefinedtefan Stănciulescu, and Thorsten Berger. 2019. Principles of feature modeling. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 62–73. https://doi.org/10.1145/3338906.3338974

[19] Muath A. Obidat. 2021. Canvas Deceiver-A New Defense Mechanism Against Canvas Fingerprinting. https://www.iiisci.org/journal/pdv/sci/pdfs/SA899XU20.pdf

[20] Chenxiong Qian, Hyungjoon Koo, ChangSeok Oh, Taesoo Kim, and Wenke Lee. 2020. Slimium: Debloating the Chromium Browser with Feature Subsetting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event USA, 2020-10-30). ACM, 461–476. https://doi.org/10.1145/3372297.3417866

[21] Uwe Ryssel, Joern Ploennigs, and Klaus Kabitzsch. 2011. Extraction of feature models from formal contexts. In *Software Product Lines Conference*. https://doi.org/10.1145/2019136.2019141

[22] Takamichi Saito, Koki Yasuda, Takayuki Ishikawa, Rio Hosoi, Kazushi Takahashi, Yongyan Chen, and Marcin Zalasiński. 2016. Estimating CPU Features by Browser Fingerprinting. In *2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)* (2016-07). 587–592. https://doi.org/10.1109/IMIS.2016.108

[23] Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In *Proceedings 2019 Network and Distributed System Security Symposium* (San Diego, CA, 2019). Internet Society. https://doi.org/10.14722/ndss.2019.23155

[24] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wąsowski, and K. Czarnecki. 2011. Reverse engineering feature models. *2011 33rd International Conference on Software Engineering (ICSE)* (2011), 461–470. https://doi.org/10.1145/1985793.1985856

[25] Steven She, Uwe Ryssel, Nele Andersen, Andrzej Wąsowski, and Krzysztof Czarnecki. 2014. Efficient synthesis of feature models. *Information and Software Technology* 56, 9 (2014), 1122–1143. https://doi.org/10.1016/j.infsof.2014.01.012 Special Sections from "Asia-Pacific Software Engineering Conference (APSEC), 2012" and " Software Product Line conference (SPLC), 2012".

[26] Anatoly Shusterman, Zohar Avraham, Eliezer Croitoru, Yarden Haskal, Lachlan Kang, Dvir Levi, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. 2021. Website Fingerprinting Through the Cache Occupancy Channel and its Real World Practicality. *IEEE Transactions on Dependable and Secure Computing* 18, 5 (2021), 2042–2060. https://doi.org/10.1109/TDSC.2020.2988369

[27] Konstantinos Solomos, Panagiotis Ilia, Soroush Karami, Nick Nikiforakis, and Jason Polakis. 2022. The dangers of human touch: fingerprinting browser extensions through user actions. In *31st USENIX Security Symposium (USENIX Security 22)*. 717–733. https://www.usenix.org/conference/usenixsecurity22/presentation/solomos

[28] Oleksii Starov and Nick Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017-05). 941–956. https://doi.org/10.1109/SP.2017.18 ISSN: 2375-1207.

[29] Jacob Swanson, Myra B. Cohen, Matthew B. Dwyer, Brady J. Garvin, and Justin Firestone. 2014. Beyond the rainbow: self-adaptive failure avoidance in configurable systems. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 377–388. https://doi.org/10.1145/2635868.2635915

[30] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. {Fp-Scanner}: The Privacy Implications of Browser Fingerprint Inconsistencies. In *27th USENIX Security Symposium (USENIX Security 18)*. 135–150.

[31] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. Fp-stalker: Tracking browser fingerprint evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 728–741. https://doi.org/10.1109/SP.2018.00008

[32] Antoine Vastel, Walter Rudametkin, and Romain Rouvoy. 2018. FP-TESTER: Automated Testing of Browser Fingerprint Resilience. In *IWPE 2018 - 4th International Workshop on Privacy Engineering* (London, United Kingdom, 2018-04) *(Proceedings of the 4th International Workshop on Privacy Engineering (IWPE'18))*. 1–5. https://hal.inria.fr/hal-01717158

[33] Nathan Weston, Ruzanna Chitchyan, and Awais Rashid. 2009. A framework for constructing semantically composable feature models from natural language requirements. In *Software Product Lines Conference*. https://dl.acm.org/doi/10.5555/1753235.1753265

[34] Yi Xiang, Xiaowei Yang, Han Huang, Zhengxin Huang, and Miqing Li. 2022. Sampling configurations from software product lines via probability-aware diversification and SAT solving. *Automated Software Engineering* 29, 2 (2022), 54. https://doi.org/10.1007/s10515-022-00348-8

[35] Sebastian Zimmeck, Jie S Li, Hyungtae Kim, Steven M Bellovin, and Tony Jebara. 2017. A privacy analysis of cross-device tracking. In *26th USENIX Security Symposium (USENIX Security 17)*. 1391–1408. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/zimmeck