BrowserFM: A Feature Model-based Approach to Browser Fingerprint Analysis

Maxime Huyghe, Clément Quinton, Walter Rudametkin

BrowserFM: A Feature Model-based Approach to Browser Fingerprint Analysis

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

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

Walter Rudametkin
Univ. Rennes, Inria, CNRS,
UMR 6074 IRISA
walter.rudametkin@irisa.fr

***Abstract*—Web browsers have become complex tools used by billions of people. The complexity is in large part due to its adaptability and variability as a deployment platform for modern applications, with features continuously being added. This also has the side effect of exposing configuration and hardware properties that are exploited by browser fingerprinting techniques.**

In this paper, we generate a large dataset of browser fingerprints using multiple browser versions, system and hardware configurations, and describe a tool that allows reasoning over the links between configuration parameters and browser fingerprints. We argue that using generated datasets that exhaustively explore configurations provides developers, and attackers, with important information related to the links between configuration parameters (*i.e.*, browser, system and hardware configurations) and their exhibited browser fingerprints. We also exploit Browser Object Model (BOM) enumeration to obtain exhaustive browser fingerprints composed of up to $16,000$ attributes.

We propose to represent browser fingerprints and their configurations with feature models, a tree-based representation commonly used in Software Product Line Engineering (SPLE) to respond to the challenges of variability, to provide a better abstraction to represent browser fingerprints and configurations. With translate $89,486$ browser fingerprints into a feature model with $35,857$ nodes from $1,748$ configurations. We show the advantages of this approach, a more elegant tree-based solution, and propose an API to query the dataset. With these tools and our exhaustive configuration exploration, we provide multiple use cases, including differences between headless and headful browsers or the selection of a minimal set of attributes from browser fingerprints to re-identify a configuration parameter from the browser.

I. Introduction

Web browsers have become ubiquitous tools, deployed across a wide spectrum of applications and use cases. They offer an extensive array of functionalities and maintain compatibility with numerous devices and peripherals. Users can customize their browsing experience through various configuration options, including settings, flags, command line switches and a diverse ecosystem of extensions. This wealth of customization options is known to render browser instances distinctly unique and identifiable through the aggregation of distinct attributes used to create unique identifiers, commonly referred to as a browser fingerprint [1]. Fingerprints are often unique because web browsers disclose diverse attributes of their computing environments, including hardware specifications [2], [3], [4], operating system parameters, installed fonts [5], browser extensions [6], [7], among other technical characteristics.

Browser fingerprint tracking can compromise user privacy. The ability to identify unique browser configurations enables sophisticated tracking methods that persist even when users attempt to maintain anonymity through conventional means, such as private browsing modes or cookie deletion. The granularity of fingerprinting attributes—ranging from hardware specifications to software configurations—can uniquely identify users across different websites and sessions [1], [8], [3], [9]. Of particular concern is the high entropy of certain attributes, which makes them especially effective. Developers and privacy advocates face the challenging task of identifying and mitigating these discriminating characteristics while maintaining existing browser features. This challenge is further complicated by the rapid evolution of browser technologies and the continuous emergence of new fingerprinting vectors, necessitating automated approaches for comprehensive identification and assessments.

We argue that feature models, a compact tree-based representation of variability in software used by the community of Software Product Line Engineering (SPLE) [10], can provide developers with abstractions and tools to better reason about browser fingerprints and to better identify and understand side-effects that lead to "fingerprintable" privacy issues. Feature models allow representing in a structured way the links between features with their constraints. Because browser fingerprints reflect the software and hardware variability of a system and are constructed through designed APIs, we have found it straightforward to represent fingerprints as feature models, allowing for a more elegant, efficient and lossless representation of large browser fingerprint datasets.

We propose to explore the links between browser fingerprints and the underlying configurations (*i.e.*, browser, system and hardware) that make them unique. Our approach proposes that developers explore browser configurations (*e.g.*, switches, flags, settings) as exhaustively as possible, collecting a browser fingerprint per configuration option in order to identify even

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2025
28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-9-0
https://dx.doi.org/10.14722/madweb.2025.23xxx
www.ndss-symposium.org

minute changes. Our approach relies on collecting exhaustive browser fingerprints through Browser Object Model (BOM) enumeration [11], which we have found may contain up to 16 thousand attributes. Furthermore, we propose to not only represent browser fingerprints as feature models, but also the browser's configuration and the hardware configuration. By including extensive system and hardware configuration information in a unified system, developers can use our approach to identify links between attributes of a browser fingerprint and the hardware, software or browser configuration that effected it.

In this paper, we present benefits of representing browser fingerprints, and the underlying configurations, as feature models, providing an elegant framework for reasoning. We also provide an API to interact with the system and we present a methodology for identifying a minimal set of attributes necessary to identify partial browser configurations from their browser fingerprints. Our approach is validated using a dataset we have constructed with $89,486$ browser fingerprints, collected from $1,748$ unique configurations parameters across 13 Chromium versions, from two hardware platforms.

The remainder of this paper is structured as follows. Section II provides background knowledge to better understand this work, in particular Section II-B presents feature modeling and their advantages in the browser fingerprint field. Section III motivates our work with practical scenarios. Section IV present our approach of representation of the browser fingerprint as feature models. Section V proposes queries to explore the browser fingerprint and configuration feature models. Section VI Section VII presents related work. Finally, Section VIII concludes the paper.

II. Background

Web browsers have evolved into highly complex software systems serving billions of users globally. Their continuous evolution is driven by the need to accommodate emerging web technologies and user requirements. Browser vendors enhance functionality through both in-house development and integration of third-party components [12], leading to increasingly sophisticated and complex codebases. This evolution has spawned a diverse ecosystem of browser implementations, each offering distinct feature sets and capabilities. The proliferation extends beyond traditional desktop browsers to include specialized variants for mobile devices, ARM-based architectures, and various operating systems. Furthermore, users can extensively customize their browsing environment through extensions, settings, experimental flags, and over a thousand switches.[1] These layers of complexity from core browser implementations to user level customizations create unique browser configurations. Each configuration variant potentially influences the browser's fingerprint, creating distinctive signatures that reflect both the underlying browser architecture and user customizations.

[1]https://peter.sh/experiments/chromium-command-line-switches/

A. Browser Fingerprinting

Browser fingerprinting employs various technical approaches and methodologies to identify unique browser configurations [13]. Some fingerprinting techniques focus on specific attribute values, such as `Canvas` rendering [14], [15] and the `User Agent` string [13], which are particularly effective for unique identification. Our research utilizes Browser Object Model (BOM) enumeration, as first presented by Schwarz *et. al* [11] to construct comprehensive browser fingerprints. BOM enumeration systematically explores the browser's public APIs to obtain all exposed attributes, values and methods from the browser, providing a comprehensive snapshot of the browser and its configuration. A significant privacy concern stems from the accessibility of these fingerprinting attributes through standard browser APIs, which do not require explicit user consent, enabling the stealthy collection of fingerprinting data. More generally, fingerprints have been shown to uniquely identify browsers without relying on cookies or logins information [1], [16]. As a result, browser fingerprinting is increasingly adopted by websites for tracking and user identification [17], and interestingly, for bot detection [18], [19].

Browser fingerprints generally consist of structured attribute-value pairs that characterize browser properties or, quite similarly, of function calls and their return values. State-of-the-art research shows that modern browser fingerprints can include over $13,000$ distinct attributes [11]. This figure continues to grow as browsers evolve and expand their functionality, we have found upwards of $16,000$ attributes, highlighting the growing complexity of fingerprinting and its implications for user privacy.

B. Encoding Software Variability

Modern Web browsers have evolved into highly-variable software systems. This variability arises from user personalization options (*e.g.*, extensions, settings, plugins), operating system compatibility (*e.g.*, Windows, macOS, Linux, Android, iOS), and diverse hardware platforms (*e.g.*, desktop computers, mobile devices, vehicles, IoT devices). This multi-dimensional variability creates an extensive space of configurations, which impacts browser fingerprint characteristics, creating a complex mapping between configuration parameters and fingerprint attributes. This complexity necessitates robust analytical tools to model and understand these configuration-fingerprint relationships.

Software Product Line Engineering provides an approach to managing variability in software systems [20]. In SPLE, feature models [21] are a commonly used tool to represent software variability. Feature models employ a hierarchical tree structure consisting of a root node and its descendant nodes. In this structure, each node may possess multiple children but is restricted to a single parent, with terminal nodes designated as leaves. For a feature model to be complete, each feature must be associated with a hierarchical constraint. The most restrictive constraint is `mandatory`, which implies that the feature is always present in every configuration if the parent feature is also present. In contrast, the least restrictive

constraint is `optional`, indicating that the feature may or may not be present if the parent feature is also present. In addition, group relationships can also be defined, such as `OR` and `XOR` relationships. In the former case, a parent feature includes one or more of its child features, while in the latter, a parent feature includes exactly one of its child features.

III. On Capturing Fingerprints Variability

Representing browser fingerprints as feature models provides a structured and systematic approach to understanding and analyzing the variability and uniqueness of browser fingerprints, as well as their practical implication.

A. Browser fingerprint sampling

A key challenge in browser fingerprint research lies in the limited access to real-world fingerprint data due to privacy concerns. Requests to share such datasets are often refused as they pose significant risks of exposing sensitive user information, limiting the ability to conduct extensive research or develop novel fingerprinting techniques. Feature model-based sampling offers a promising solution to this issue [22]. By using the constraints and variability encoded in the feature model, one can generate synthetic browser fingerprints that are both realistic and diverse. That is, synthetic fingerprints are derived from a feature model based on an existing dataset. These synthetic fingerprints retain the structural characteristics and variability of real-world data without exposing the original, sensitive data. Moreover, even a small dataset of browser fingerprints can be significantly expanded through sampling techniques by systematically exploring the variability within the feature model and generating new fingerprints.

B. User identification

Browser Object Model (BOM) enumeration collects thousands of attributes per fingerprint, significantly increasing the overall fingerprint size compared to other more targeted approaches. The collection and comparison of browser fingerprints during each website visit presents substantial data management and storage challenges. While various techniques exist for user re-identification across multiple fingerprints[2], including full fingerprint hashing and partial attribute subset hashing, these approaches have inherent limitations. Although hashing provides efficient user re-identification, it proves fragile in practice, making it difficult to identify specific changes when hash values differ between visits. Hence, the selection of the attributes to create the hashed value must target those with high entropy that remain stable over extended periods of time. Furthermore, the redundancy of attributes across multiple browser fingerprints presents an additional challenge. The repetitive storage of identical attributes across multiple browser fingerprints introduces unnecessary computational overhead and storage inefficiencies, suggesting the need for an optimized data management strategy. These limitations in current approaches highlight the need for an innovative fingerprint representation that optimizes storage requirements while maintaining reliable identification capabilities and an better comprehension of the browser fingerprint at a large scale.

C. Evolution and History

Browser fingerprint generation always occurs client-side, while storage, analysis, and comparison are typically performed server-side. This is common practice because any client-side calculations can be spoofed and are thus better protected server-side. This architectural separation poses challenges for researchers, as it make the fingerprinting process difficult to identify [23] and limits access to the fingerprint datasets and the evolution of fingerprints over time. Previous efforts to address the challenge of understanding fingerprints overtime have relied on browser extensions to collect fingerprints from the same devices over long periods of time [8], [3]. While such solutions provide some utility, they also exhibit several limitations. For instance, extensions such as AmIUnique[3,4] provide users access only to their individual fingerprints but not to the entire dataset because it can be used to re-identify other users and may contain other sensitive data. Such an approach does not support the analysis of global fingerprint patterns or variations over time by the larger research community. Furthermore, data collected via such tools is inherently biased, as users who install privacy-focused extensions typically exhibit higher privacy awareness and often utilize additional privacy-enhancing technologies, making them unrepresentative of the general population [24]. Finally, without access to high-traffic websites, it remains challenging to collect a substantial and representative dataset of browser fingerprints, especially one that spans over a long period. And in any case, browsers protect the device and limit what websites can collect. The browser's configuration, the system configuration, and the hardware's configuration and specifics can not be collected through JavaScript, making any inferences difficult or limited because of the lack of ground truth values.

We propose an alternative to these website-based datasets, which contain information that may re-identify users and are thus not shareable nor otherwise reviewable by users or researchers. By exhaustively exploring browser configurations and collecting fingerprints on our own hardware, we can generate datasets that better understand the fingerprinting surface of browsers, the links between configuration parameters and fingerprinting attributes, and we can share the datasets freely since they no longer re-identify other users. The tradeoff is a loss of the extensive diversity of devices found in other datasets [1], [16], [24] for a gain in controlling and collecting the entire hardware, system and browser configurations. Our approach relies on exhaustively exploring browser configurations (*i.e.*, browser version, switches, flags and settings) on hardware in our control for which we also collect extensive

[2]https://github.com/fingerprintjs/fingerprintjs

[3]https://addons.mozilla.org/firefox/addon/amiunique/

[4]https://chrome.google.com/webstore/detail/amiunique/pigjfndpomdldkmoaiiigpbncemhjeca

configuration information. Furthermore, by representing fingerprints as feature models, existing approaches on feature model and software product line evolution [25], [26] can thus be applied. For example, by constructing a feature model for each browser version, it becomes easy to compare how and when browser features are introduced or deprecated, track changes across different browser versions, and identify attributes with high entropy.

D. Fingerprint Storage

During our experiments, we compared the size of browser fingerprints stored in a traditional format, such as a `json` file, with their representation in our feature model. For a set of $89,486$ browser fingerprints, the `json` representation occupies 121 GB, whereas the feature model requires only 13.4 GB, resulting in a size reduction of 9 times. It is worth noting that our feature models include some additional metadata not common in other modeling approaches. Figure 1 shows that each and every node stores the UUIDs of the fingerprints that contained that node (see Section IV-B). This makes the conversion to the feature model lossless, as well as allowing for various queries (see Section V). However, some use cases, such as fingerprint sampling, do not require the additional metadata. When removed, the size of the feature model further decreases to 2.1 MB. This significant reduction enables long-term storage of a greater number of browser fingerprints, facilitating a more comprehensive understanding of their characteristics and evolution over time.

E. Browser Comparison

There is a wide variety of web browsers. However, the majority of popular browsers are currently derived from Chromium. Some browsers, such as *Brave*[5], *Ungoogled Chromium*[6], and *Epic Privacy Browser*[7], place a strong emphasis on privacy. Two of these browsers specifically claim to protect users against browser fingerprinting through various techniques. By encoding browser fingerprints in feature models, we can facilitate comparisons between Chromium and its derivatives to then evaluate the effectiveness of their fingerprinting protection mechanisms. We can compare these browsers to gain a deeper understanding of how their differences, and more specifically, their anti-fingerprinting tools impact browser fingerprints.

F. Reduction of Fingerprinting Detection

Recent research in anti-fingerprinting technologies has focused extensively on analyzing attribute sets targeted by fingerprinting scripts [27], [28], [29], [23]. Understanding the complex relationships between browser configurations and their corresponding attributes is crucial for developing sophisticated fingerprint algorithms as well as optimal spoofing countermeasures. This knowledge enables the implementation of dynamic attribute selection mechanisms that can vary with each browser session, effectively enhancing privacy protection. By identifying minimal subsets of attributes necessary for re-identification, we can strongly reduce the attributes needed as well as the time to fingerprint a device. Furthermore, we find that many attributes are either activated or deactivated in groups when a configuration parameter changes, making the attributes have similar importance in identifying the browser or the configuration parameter. For example, WebGL support adds about 2000 attributes simultaneously to the BOM. Instead of collecting all WebGL-related attributes, the system may verify the presence of WebGL support and authenticate the browser version, preventing unnecessary attribute collection while maintaining robust spoofing detection capabilities due to the many similar attributes that can be tested. Identifying these attributes and selecting minimal sets can make the fingerprinting process much more efficient. Also, by implementing randomized attribute selection strategies on attributes with similar levels of importance, we can create dynamically generated fingerprinting scripts that can mitigate client-side attribute spoofing in a moving target approach. This targeted approach significantly improves both the efficiency and the effectiveness in detecting spoofed browser configurations.

IV. Fingerprints as Feature Models

The representation of browser fingerprints naturally maps to the hierarchical structure of feature models. This is arguably because the BOM exposes the browser's APIs, which are designed in a structured manner as any API is. We take each attribute from the fingerprint, seen in it's fully qualified path form (*e.g.*, `window.screen.width`, and decompose it into a sequence of nodes called features in SPLE (*e.g.*, `window`, `screen`, `width`), beginning from the root node (*i.e.*, `window` for all browser fingerprints) and extending through each path component. In our model, attribute values are represented as leafs at the termination points of their respective branches. This structure maintains a one-to-one relationship between an attribute and its value within a single fingerprint, while allowing for the aggregation of different values across multiple fingerprints. Consequently, while an attribute path may be associated with multiple leaf nodes when considering different fingerprints, it maintains strict uniqueness within the context of any individual fingerprint. An example can be seen in the *Fingerprints* feature model presented in Figure 1.

A. Generating Browser Fingerprints and Linking Configurations

To construct our feature model[8], we generate $89,486$ browser fingerprints using the Chromium browser from versions *115.0.5790* to *127.0.6533*. We run each browser version thousands of times to collect a fingerprint for unique browser configurations. We also explore the browsers in both headless and headful configurations, an interesting use case to identify bots. During each launch, we record the browser's

[5]https://brave.com

[6]https://github.com/ungoogled-software/ungoogled-chromium

[7]https://epicbrowser.com/

[8]https://s.42l.fr/BrowserFM

configuration, and importantly, its environment, which includes the system's hardware and software information. By representing the browser fingerprint within the feature model, a framework traditionally used to represent configurations, we extend its application. Specifically, we create a new feature model based on the configurations and environments employed during browser fingerprint generation. Ultimately, we obtain three distinct feature models, each representing a different aspect of the process:

- The browser fingerprint feature model, encompassing $89,486$ distinct fingerprints.
- The system information feature model, including the hardware and software characteristics of the system in which the browser was executed during fingerprint generation.
- The browser's configuration, detailing the browser version and its configuration parameters.

These three feature models enable us to determine which configurations impact the browser's fingerprint and the nature of their impact. Leveraging this understanding, we can identify the attributes in the browser fingerprint that are affected by specific configurations and use this information to link a given fingerprint and re-identify its originating configuration parameter.

B. Building feature models

A browser fingerprint can be formally represented as a hierarchical structure of attribute-value pairs. We construct a tree-based representation according to the following transformation rules:

- Each distinct attribute is transformed into an internal node in the tree structure;
- The corresponding values are represented as leaf node in the hierarchy;
- Nested attributes establish parent-child relationships, where containing attributes become parent node to their constituent attributes.

Like mentioned in section II-B, we use constraints to enhance our understanding of the relationships among subsets of attributes, however, they do not establish relationships between different branches of the feature model. To address this limitation, we assign a unique `UUID` to each generated fingerprint and associate it directly with each feature. This approach allows us to trace which attributes are influenced by specific configurations or environments, as each `UUID` is also stored in the nodes of the system information and browser information feature models. An additional advantage of this approach is the ability to reconstruct the original browser fingerprint from the feature model, even when the feature model contains a large collection of fingerprints. In the end, all browser fingerprints are consolidated into a single feature model. Unlike the case where all fingerprints would be merged into a single `JSON` file—resulting in the loss of their origins, this approach preserves the provenance of each fingerprint, eliminates redundancy, and provides a more structured storage solution. We constructed three feature models. The first is based on $89,486$ browser fingerprints and contains $35,857$ nodes with $18,194$ leafs (values). The second represents system information, with 464 nodes. The third captures browser information, comprising $92,998$ nodes. These three feature models enable us to directly track configurations and their impacts on browser fingerprints, and provide to us different advantages and use cases. Figure 1 illustrates a partial representation of the three feature models, showing how information (including names, constraints, children, and `UUIDs`) is stored within each feature. The example showcases six browser fingerprints, four of them are generated from a browser with the `disable-3d-apis` switch enabled. Their corresponding `UUIDs` appear in the `disable-3d-apis` feature of the BrowserInformation model but are notably absent from the `wgl` feature in the Fingerprints model. Such absence shows how enabling this switch affects the browser fingerprint by preventing certain features from being present, and demonstrates the direct impact of specific configurations on browser fingerprints. To ease practical use of our approach, we propose an API, allowing for seamless queries across these feature models to efficiently extract the desired data.

V. Querying the Feature Model

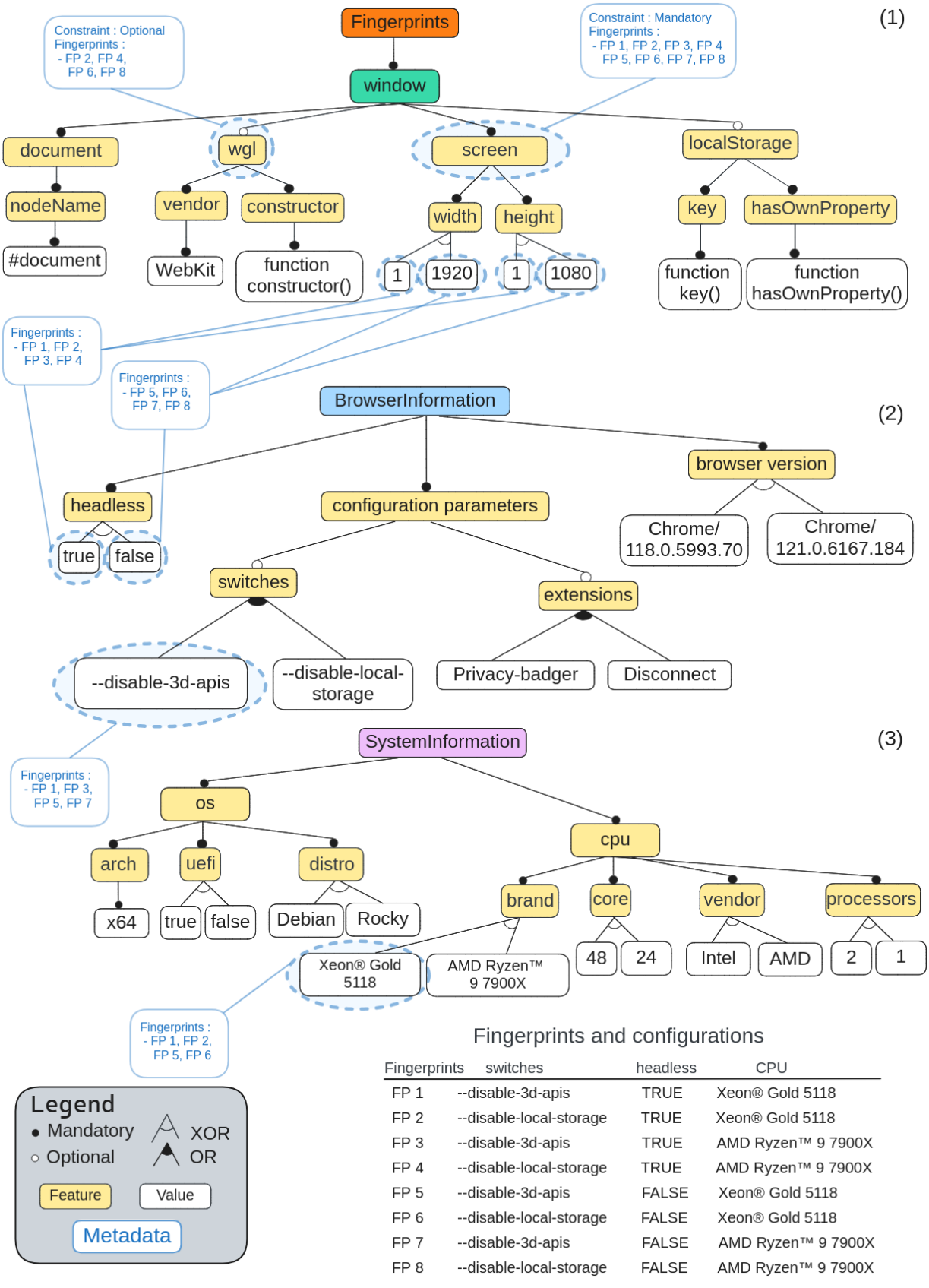
To leverage the feature model, we developed an API for querying fingerprint information. This API includes, but is not limited to, queries for :

- Fingerprint reconstruction: Reconstruct a complete fingerprint from its attributes.
- Fingerprint retrieval: Retrieve a fingerprint based on a specific criteria (*e.g.*, `UUID`).
- Configuration identification: Identify all configurations where a particular attribute is present.

An excerpt of this API is shown in Listing 1, where three queries are computed on a feature model built from a JSON file. In the following, we provide more details on three core queries of the API.

Extracting fingerprint. To reconstruct complete fingerprints from their constituent attributes, the process traverses the feature tree for identifying nodes that contain the target `UUID` identifier. When a feature node contains the specified `UUID`, it is stored in memory, and the algorithm continues its traversal through the node's children. For nodes that do not contain the target `UUID`, the algorithm examines sibling nodes. If no siblings are present, the search continues with the parent node's siblings. Upon completion of the tree traversal, all features and their hierarchical paths associated with the target `UUID` are assembled into a complete fingerprint structure. This reconstructed fingerprint can then be serialized into various formats, such as JSON or key-value pairs, depending on the specific implementation requirements.

Searching attributes. The process of extracting attributes based on specific constraints begins with the definition of search criteria, such as targeting a particular browser version. The algorithm first queries the browser information feature model to identify all `UUIDs` associated with the specified



Fingerprints and configurations

Fingerprints	switches	headless	CPU
FP 1	--disable-3d-apis	TRUE	Xeon® Gold 5118
FP 2	--disable-local-storage	TRUE	Xeon® Gold 5118
FP 3	--disable-3d-apis	TRUE	AMD Ryzen™ 9 7900X
FP 4	--disable-local-storage	TRUE	AMD Ryzen™ 9 7900X
FP 5	--disable-3d-apis	FALSE	Xeon® Gold 5118
FP 6	--disable-local-storage	FALSE	Xeon® Gold 5118
FP 7	--disable-3d-apis	FALSE	AMD Ryzen™ 9 7900X
FP 8	--disable-local-storage	FALSE	AMD Ryzen™ 9 7900X

Fig. 1. Partial representation of the three feature models. (1) the Browser Fingerprint model capturing attribute characteristics. (2) the Browser Information model containing browser configuration and version. And (3) the System Information model representing hardware and software specifications.

```
browser_query = FeatureModelQuery(
    model=FeatureModelLoader.load("path/to/browserModel.pkl"),
    model_type=FeatureModelType.BROWSER_INFO
)
fingerprint_query = FeatureModelQuery(
    model=FeatureModelLoader.load("path/to/fingerprintModel.pkl"),
    model_type=FeatureModelType.FINGERPRINT
)
system_query = FeatureModelQuery(
    model=FeatureModelLoader.load("path/to/systemModel.pkl"),
    model_type=FeatureModelType.SYSTEM_INFO
)

fingerprint = fingerprint_query.from_origin({'123e4567-e89b-12d3-a456-426614174000'}) \
                               .get_attributes_values_json() \
                               .print_results()

browser_version_uuids = browser_query.from_name({'Chrome/125.0.6422.141'}) \
                                     .get_uuids() \
                                     .print_results()

attributes = fingerprint_query.from_origin({browser_version_uuids}) \
                              .from_name({'languages','0','en_us'}) \
                              .get_attributes_values() \
                              .print_results()

configurations = system_query.from_name({'displays','Display_1','model'}) \
                             .get_children() \
                             .get_uuids() \
                             .print_results()

```

Listing 1: API Requests for extracting a fingerprint and searching for attributes and configurations

browser version. These UUIDs then serve as input for exploring the broader browser fingerprint feature model, following the previously described traversal methodology. The constraint-based search can be further refined by incorporating additional criteria such as timezone, language preferences, or screen resolution specifications. These additional constraints are applied sequentially, filtering the attribute set while maintaining the initial browser version constraint. The result is a precise subset of attributes that satisfies all specified constraints across both feature models.

Searching configurations. Understanding attribute presence patterns provides valuable insights into configuration-dependent behaviors. By analyzing when specific attributes appear or disappear based on system configurations, we can identify direct relationships between hardware, operating system or browser configurations and fingerprint characteristics. For example, observing the correlation between dedicated graphics card presence and certain attributes helps establish clear hardware-fingerprint dependencies. This knowledge not only enhances our understanding of fingerprint generation but also creates potential pathways for identifying virtual environments or automated systems through attribute analysis. Such insights could prove particularly valuable for future research in bot detection and virtual machine identification.

VI. Minimal fingerprints

One interesting application of feature models is determining the minimal set of attributes in a browser fingerprint required to identify specific configuration elements. To achieve this, we selected certain sets of switches based on a subset of attributes. We established links between attributes and switches, assigning a weight to each attribute. The weight was determined by the number of switches affecting the attribute for a given browser version. Specifically, if an attribute was impacted by only one switch, it was considered highly important. Conversely, if an attribute was influenced by multiple switches, its weight decreased proportionally to the number of switches affecting it. We retrieve this information by counting the UUIDs in the feature model. In our experiment, we first generated a sample of browser fingerprints using various switches, as detailed in section IV-A. We measured and recorded the impact of each switch on the browser fingerprints. Subsequently, we generated a new sample on a different machine with a distinct set of switches known to affect browser fingerprints The goal was to identify these switches based on their impact on the browser fingerprint. During the identification process, we analyzed the fingerprints impacted by switches and successfully identified between 74.7% and 87.3% of the switches. This experiment was repeated with progressively smaller subsets of attributes ranked by weight: the top 75%, 50%, 25%, 10%, and finally just one attribute (the highest-weighted) for

the 0% case. For the 0% case, only the single most critical attribute was used to identify switches. During the experiment, the algorithm was provided with the browser version, the fingerprint, a list of attributes with their weights, and the links between attributes and switches. Based on this information, the algorithm deduced the list of switches that could plausibly have impacted the fingerprint. Feature models proved instrumental in identifying minimal attribute sets necessary for detecting configuration elements in browser fingerprints. By leveraging weighted attribute analysis, where weights are inversely proportional to the number of switches influencing each attribute, this approach prioritizes attributes uniquely tied to specific switches over those impacted by multiple switches. Figure 2 highlights two significant changes in identification performance: one between 0% and 10%, and another between 25% and 75%, depending on the browser version. Notably, it is surprising to observe that identification rates range from 63.1% to 76.4% using only a single attribute. These results demonstrate that certain attributes carry high entropy and are particularly effective for configuration identification. From the data presented in Figure 2, we can infer that efficient configuration identification can be achieved with as little as 10% of the impacted attributes. Additionally, the performance gains between 75% and 100% attribute selection vary depending on the browser version. The minimal browser fingerprint can compromise the entropy for user re-identification, as it focuses on identifying the client-side configuration rather than directly identifying the user. However, in future work, we aim to refine this approach by selecting the optimal set of attributes for user identification. By adjusting attribute selection, we can achieve a balanced trade-off between identification accuracy and performance.

VII. Related Work

A. Software variability

In the context of software variability, Swanson et al. [30] propose a methodology to track valid configurations over time and provide solutions when inconsistencies arise. This represents a crucial tool for evolving software systems, where configuration changes may lead to operational errors. Several studies have explored the application of Software Product Line Engineering techniques in diverse domains. For instance, Cashman et al. [31] demonstrate the potential of feature models in biological applications. Specifically, they show how feature models facilitate the reuse of DNA elements within their `Biobrick` repository. These studies highlight both the versatility of feature models across different domains and their potential applications to browser fingerprinting scenarios. The feature model has different constraints, and the state of the proposal different approach. She et al. [32] tackled the problem with automatic model synthesis from propositional constraint on two normal form (conjunctive and disjunctive), providing an NP-hardness enhancement. Ryssel et al. [33] propose to incorporate the concept of or-groups and xor-groups. Their proposal can automatically extract the relation between features from existing configurations. Acher et al. [34] have an approach to extract the feature model from the product description, then permit to have a model process more structured and efficient. The study of Herrejon et al. [35] propose an application of evolutionary algorithms for reverse engineer a feature models. Their approach provides the potential to reconstruct a feature models from existing systems.

B. Browser Fingerprinting

Schwarz et al. [11] propose a framework for BOM enumeration, the creation of templates for browser fingerprints and their comparison. However, in their framework, the cleaning phase is based on the difference after a refresh of the web page, and doesn't include the other dynamic attributes in longer terms. During the analysis phase, they compare browser fingerprints one by one, which is not scalable at large scale and limits the usage. Huyghe et al. [36] propose an approach to represent browser fingerprints in feature models, however they limit their usage to browser fingerprints, don't provide any ways to explore them and don't share any results on the identification of configurations from browser fingerprints. Andriamilanto et al. [37] propose a tool to select the best attributes to identify users. However, the experimentation is limited to a handful of attributes, and doesn't take into consideration all other existing attributes with potentially higher entropy.

VIII. Conclusion

This work highlights the potential of feature models in the domain of browser fingerprint analysis. Theses representation give access to a series of tools and approaches from the SPLE domain, offering new opportunities to get a better comprehension of the browser fingerprint and optimize fingerprinting defenses, including dynamic attribute selection and efficient verification processes. Our experiments demonstrate that certain attributes exhibit high entropy, making them particularly impactful for configuration identification. Moreover, we showed that meaningful insights can be derived even with a reduced set of attributes, thereby reducing redundancy and improving storage efficiency. These findings underline the importance of understanding the relationships between browser features, configurations, and fingerprints in order to enhance privacy protection mechanisms.

Acknowledgment

This work has been financially supported by the Agence Nationale de la Recherche through the ANR-21-CE39-0019 FACADES[9] and the ProjetIA-22-PECY-0002 iPoP[10] projects and was made possible by Software Heritage[11], the great library of source code.

[9]https://anr.fr/Project-ANR-21-CE39-0019
[10]https://anr.fr/ProjetIA-22-PECY-0002
[11]https://www.softwareheritage.org/

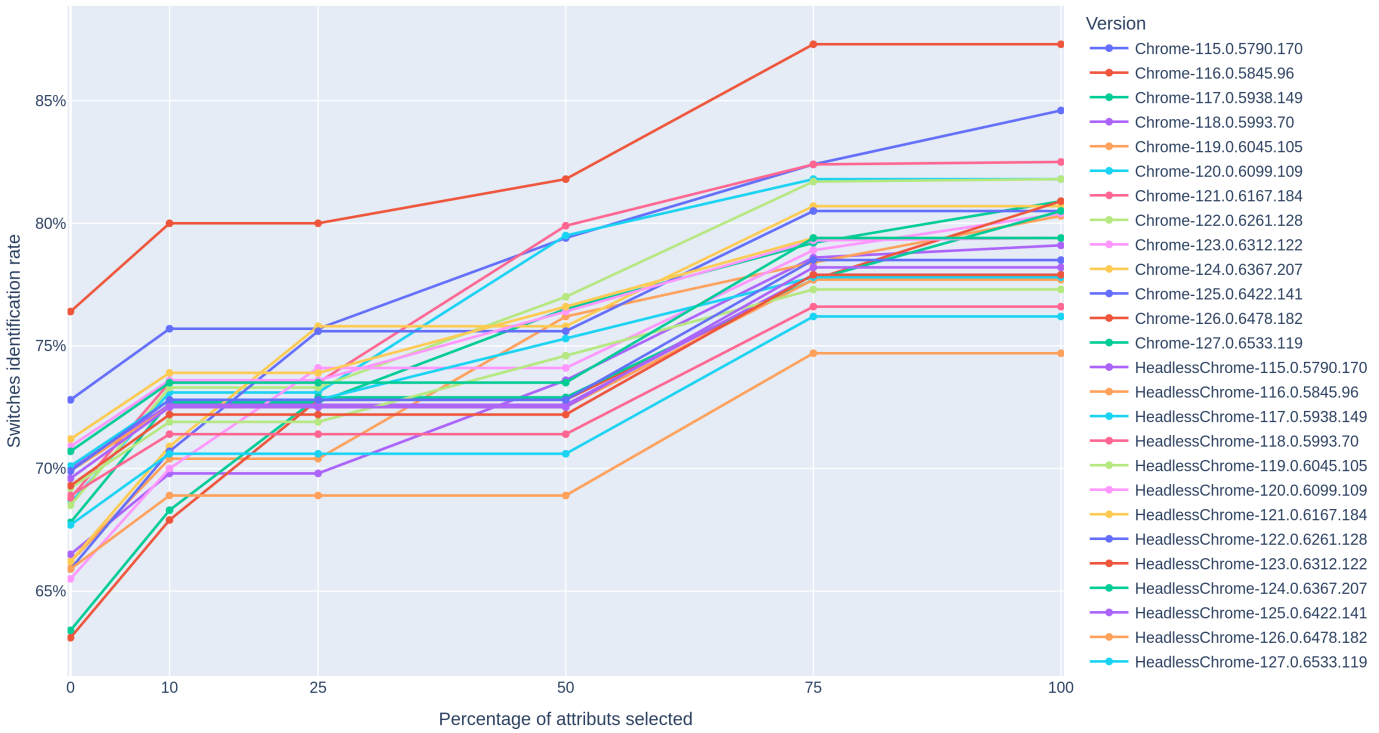


Fig. 2. Identification results by browser version, showing the percentage of selected attributes (ranging from a single attribute at 0% to 100%)and their effectiveness in identifying switches.

REFERENCES

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

[2] T. Saito, K. Yasuda, T. Ishikawa, R. Hosoi, K. Takahashi, Y. Chen, and M. Zalasiński, "Estimating CPU features by browser fingerprinting," in *2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 2016, pp. 587–592. [Online]. Available: https://doi.org/10.1109/IMIS.2016.108

[3] T. Laor, N. Mehanna, A. Durey, V. Dyadyuk, P. Laperdrix, C. Maurice, Y. Oren, R. Rouvoy, W. Rudametkin, and Y. Yarom, "DRAWN APART : A device identification technique based on remote GPU fingerprinting," in *Proceedings 2022 Network and Distributed System Security Symposium*. Internet Society, 2022.

[4] S. Chalise, H. D. Nguyen, and P. Vadrevu, "Your speaker or my snooper? measuring the effectiveness of web audio browser fingerprints," in *Proceedings of the 22nd ACM Internet Measurement Conference*, ser. IMC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 349–357. [Online]. Available: https://doi.org/10.1145/3517745.3561435

[5] D. Fifield and S. Egelman, "Fingerprinting web users through font metrics," in *Financial Cryptography and Data Security*, R. Böhme and T. Okamoto, Eds. Springer Berlin Heidelberg, 2015, vol. 8975, pp. 107–124, series Title: Lecture Notes in Computer Science. [Online]. Available: http://doi.org/10.1007/978-3-662-47854-7_7

[6] A. Vastel, W. Rudametkin, and R. Rouvoy, "FP-TESTER: Automated testing of browser fingerprint resilience," in *IWPE 2018 - 4th International Workshop on Privacy Engineering*, ser. Proceedings of the 4th International Workshop on Privacy Engineering (IWPE'18), 2018, pp. 1–5. [Online]. Available: https://hal.inria.fr/hal-01717158

[7] O. Starov and N. Nikiforakis, "XHOUND: Quantifying the fingerprintability of browser extensions," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 941–956, ISSN: 2375-1207. [Online]. Available: https://doi.org/10.1109/SP.2017.18

[8] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy, "Fp-stalker: Tracking browser fingerprint evolutions," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 728–741. [Online]. Available: https://doi.org/10.1109/SP.2018.00008

[9] M. Huyghe, C. Quinton, and W. Rudametkin, "FP-Rainbow: Fingerprint-Based Browser Configuration Identification," in *Proceedings of the ACM Web Conference 2025, WWW'25*, Sydney, Australia, 2025, pp. 1–11.

[10] C. Quinton, L. Duchien, P. Heymans, S. Mouton, and E. Charlier, "Using feature modelling and automations to select among cloud solutions," in *2012 Third International Workshop on Product LinE Approaches in Software Engineering (PLEASE)*, 2012, pp. 17–20.

[11] M. Schwarz, F. Lackner, and D. Gruss, "JavaScript template attacks: Automatically inferring host information for targeted exploits," in *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, 2019. [Online]. Available: https://doi.org/10.14722/ndss.2019.23155

[12] C. Qian, H. Koo, C. Oh, T. Kim, and W. Lee, "Slimium: Debloating the chromium browser with feature subsetting," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 461–476. [Online]. Available: https://doi.org/10.1145/3372297.3417866

[13] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine, "Browser fingerprinting: A survey," *ACM Transactions on the Web (TWEB)*, vol. 14, no. 2, pp. 1–33, 2020. [Online]. Available: https://doi.org/10.1145/3386040

[14] M. A. Obidat, "Canvas deceiver-a new defense mechanism against canvas fingerprinting," in *Systemics, Cybernetics and Informatics*, 2021. [Online]. Available: https://www.iiisci.org/journal/pdv/sci/pdfs/SA899XU20.pdf

[15] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in html5," *Proceedings of W2SP*, 2012. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3208feae829cba6bd319421fe1fea58962da8fd9

[16] P. Laperdrix, W. Rudametkin, and B. Baudry, "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, May 2016. [Online]. Available: https://inria.hal.science/hal-01285470

[17] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. Association for Computing Machinery, 2016, pp. 1388–1401. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978313

[18] A. Vastel, W. Rudametkin, R. Rouvoy, and X. Blanc, "FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers," in *MADWeb'20 - NDSS Workshop on Measurements, Attacks, and Defenses for the Web*, O. Starov, A. Kapravelos, and N. Nikiforakis, Eds., San Diego, United States, Feb. 2020. [Online]. Available: https://inria.hal.science/hal-02441653

[19] B. Amin Azad, O. Starov, P. Laperdrix, and N. Nikiforakis, "Web runner 2049: Evaluating third-party anti-bot services," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*. Springer, 2020, pp. 135–159.

[20] A. Metzger and K. Pohl, "Software product line engineering and variability management: achievements and challenges," in *Future of Software Engineering Proceedings*, ser. FOSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 70–84. [Online]. Available: https://doi.org/10.1145/2593882.2593888

[21] D. Nešić, J. Krüger, u. Stănciulescu, and T. Berger, "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*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 62–73. [Online]. Available: https://doi.org/10.1145/3338906.3338974

[22] Y. Xiang, X. Yang, H. Huang, Z. Huang, and M. Li, "Sampling configurations from software product lines via probability-aware diversification and sat solving," *Automated Software Engineering*, vol. 29, no. 2, p. 54, 2022. [Online]. Available: https://doi.org/10.1007/s10515-022-00348-8

[23] U. Iqbal, S. Englehardt, and Z. Shafiq, "Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors," *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1143–1161, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221094754

[24] A. Gómez-Boix, P. Laperdrix, and B. 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*, 2018, pp. 309–318. [Online]. Available: https://doi.org/10.1145/3178876.3186097

[25] M. Acher, P. Heymans, P. Collet, C. Quinton, P. Lahire, and P. Merle, "Feature model differences," in *Advanced Information Systems Engineering*, J. Ralyté, X. Franch, S. Brinkkemper, and S. Wrycza, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 629–645.

[26] C. Kröher, L. Gerling, and K. Schmid, "Comparing the intensity of variability changes in software product line evolution," *Journal of Systems and Software*, vol. 203, p. 111737, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223001322

[27] A. Gómez-Boix, D. Frey, Y.-D. Bromberg, and B. Baudry, "A collaborative strategy for mitigating tracking through browser fingerprinting," in *Proceedings of the 6th ACM Workshop on Moving Target Defense*, 2019, pp. 67–78.

[28] C. F. Torres, H. Jonker, and S. Mauw, "Fp-block: usable web privacy by controlling browser fingerprinting," in *Computer Security–ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II 20*. Springer, 2015, pp. 3–19.

[29] 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*. IEEE, 2013, pp. 541–555.

[30] J. Swanson, M. B. Cohen, M. B. Dwyer, B. J. Garvin, and J. Firestone, "Beyond the rainbow: self-adaptive failure avoidance in configurable systems," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 377–388. [Online]. Available: https://doi.org/10.1145/2635868.2635915

[31] M. Cashman, J. Firestone, M. B. Cohen, T. Thianniwet, and W. Niu, "Dna as features: Organic software product lines," in *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, ser. SPLC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 108–118. [Online]. Available: https://doi.org/10.1145/3336294.3336298

[32] S. She, U. Ryssel, N. Andersen, A. Wasowski, and K. Czarnecki, "Efficient synthesis of feature models," *Information and Software Technology*, vol. 56, no. 9, pp. 1122–1143, 2014, special Sections from "Asia-Pacific Software Engineering Conference (APSEC), 2012" and " Software Product Line conference (SPLC), 2012". [Online]. Available: https://doi.org/10.1016/j.infsof.2014.01.012

[33] U. Ryssel, J. Ploennigs, and K. Kabitzsch, "Extraction of feature models from formal contexts," in *Software Product Lines Conference*, 2011. [Online]. Available: https://doi.org/10.1145/2019136.2019141

[34] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," *Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems*, 2012. [Online]. Available: https://doi.org/10.1145/2110147.2110153

[35] R. E. Lopez-Herrejon, J. A. Galindo, D. Benavides, S. Segura, and A. Egyed, "Reverse engineering feature models with evolutionary algorithms: An exploratory study," in *International Symposium on Search Based Software Engineering*, 2012. [Online]. Available: https://doi.org/10.1007/978-3-642-33119-0_13

[36] M. Huyghe, C. Quinton, and W. Rudametkin, "Taming the variability of browser fingerprints," in *Proceedings of the 28th ACM International Systems and Software Product Line Conference*, 2024, pp. 66–71.

[37] N. Andriamilanto, T. Allard, and G. Le Guelvouit, "Fpselect: low-cost browser fingerprints for mitigating dictionary attacks against web authentication mechanisms," in *Proceedings of the 36th Annual Computer Security Applications Conference*, 2020, pp. 627–642.