FPSelect: Low-Cost Browser Fingerprints for Mitigating Dictionary Attacks against Web Authentication Mechanisms

Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit

FPSelect: Low-Cost Browser Fingerprints for Mitigating Dictionary Attacks against Web Authentication Mechanisms

Nampoina Andriamilanto*
tompoariniaina.andriamilanto@irisa.fr
Univ Rennes, CNRS, IRISA
Rennes, France

Tristan Allard
tristan.allard@irisa.fr
Univ Rennes, CNRS, IRISA
Rennes, France

Gaëtan Le Guelvouit
gaetan.leguelvouit@b-com.com
IRT b<>com
Cesson-Sévigné, France

ABSTRACT

Browser fingerprinting consists into collecting attributes from a web browser. Hundreds of attributes have been discovered through the years. Each one of them provides a way to distinguish browsers, but also comes with a usability cost (e.g., additional collection time). In this work, we propose FPSelect, an attribute selection framework allowing verifiers to tune their browser fingerprinting probes for web authentication. We formalize the problem as searching for the attribute set that satisfies a security requirement and minimizes the usability cost. The security is measured as the proportion of impersonated users given a fingerprinting probe, a user population, and an attacker that knows the exact fingerprint distribution among the user population. The usability is quantified by the collection time of browser fingerprints, their size, and their instability. We compare our framework with common baselines, based on a real-life fingerprint dataset, and find out that in our experimental settings, our framework selects attribute sets of lower usability cost. Compared to the baselines, the attribute sets found by FPSelect generate fingerprints that are up to 97 times smaller, are collected up to 3,361 times faster, and with up to 7.2 times less changing attributes between two observations, on average.

CCS CONCEPTS

• **Security and privacy → Multi-factor authentication**; **Browser security**; **Web application security**.

KEYWORDS

browser fingerprinting, web authentication, multi-factor authentication

ACM Reference Format:
Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. 2020. FPSelect: Low-Cost Browser Fingerprints for Mitigating Dictionary Attacks against Web Authentication Mechanisms. In *Annual Computer Security Applications Conference (ACSAC 2020), December 7–11, 2020, Austin, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3427228.3427297

1 INTRODUCTION

Nowadays, the managers of web platforms face a crucial choice about which authentication mechanism to use. On the one hand, using solely passwords is common and easy, but fallible due to the many attacks that exist: brute force, dictionary [61], credential stuffing [57], or targeted knowledge attacks [60]. Previous studies report over 3.3 billion credentials leaked [36], a password reuse rate above 30% [20], and the risk of account hijacking increased by 400 times if the credentials of an account are stolen through a phishing attack [57]. On the other hand, using supplementary authentication factors improves security, but at the cost of usability [11]. Indeed, users are required to remember, possess, or undergo additional actions, which is impractical in real-life (e.g, using a security token requires users to constantly carry it). As a result, few users authenticate using multiple factors (e.g., it is estimated that less than 10% of the active Google accounts use two factors [36, 42]).

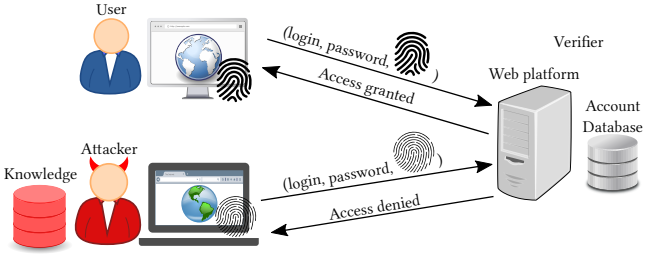


Figure 1: Example of a browser fingerprinting web authentication mechanism and a failed attack.

Initially used to track users on the web, *browser fingerprinting* has recently been identified as a promising authentication factor [3, 4, 43, 53, 54, 58]. It consists into collecting the values of attributes from a web browser (e.g., the `UserAgent` HTTP header [47], the screen resolution, the way it draws a picture [38]) to build a fingerprint. This technique is already endorsed by open source access management solutions (e.g., OpenAM[1]) and by software products (e.g., SecureAuth[2]).

The two adversarial participants are the verifier and the attacker, as depicted in Figure 1. The verifier aims to protect the users of her web platform, using an authentication mechanism based on browser fingerprinting. The verifier stores the fingerprint of the usual browser of each user. On each login, the fingerprint of the browser in use is matched against the fingerprint that is stored for the claimed account. The attacker tries to impersonate the users by submitting specially crafted fingerprints. The aim of the verifier is to limit the reach of the attacker, also called *sensitivity* below,

*Also with IRT b<>com.

[1]https://backstage.forgerock.com/docs/am/6.5/authentication-guide/#device-id-match-hints
[2]https://docs.secureauth.com/x/agpjAg

ACSAC 2020, December 7–11, 2020, Austin, USA

which is measured as the proportion of impersonated users. To do so, she builds a fingerprinting probe that integrates one or more attributes, that are selected among the hundreds[3] that have been discovered over the years [3, 13, 14, 33, 55]. On the one hand, the addition of an attribute to the probe can strengthen the distinctiveness of browsers, hence reducing the sensitivity. On the other hand, each addition comes with a *usability cost* that may render the probe impractical in an online authentication context. Indeed, each attribute consumes storage space (up to hundreds of kilobytes [12]), collection time (up to several minutes [37, 39, 40, 45, 48, 49]), and can increase the instability of the generated fingerprints [59]. For example, considering all of our attributes leads to a fingerprint taking 9.98 seconds on average to collect, which is impractical for the user. Moreover, some attributes are strongly correlated together [5], and including them only increases the usability cost without reducing the sensitivity. Due to these correlations, picking attributes one by one independently may lead to poor sensitivity and usability scores.

Previous works only consider the well-known attributes [13, 19, 33], remove the attributes of the lowest entropy [59], iteratively pick the attribute of the highest weight (typically the entropy) until a threshold is reached [8, 14, 23, 29, 35, 56], or evaluate every possible set [16]. The entropy measures the skewness of the distribution of fingerprints or attribute values. As pointed out by Acar [1], it does not take the worst cases into account (i.e., the most common values that attackers can submit similarly to dictionary attacks on passwords [10]). Moreover, fingerprints cannot be compared identically like passwords due to their evolution through time. The attackers do not need to find the exact fingerprint of a victim, but one that is similar enough to deceive the verification mechanism.

In this paper, we propose FPSelect, a framework that allows a verifier to select the attributes[4] to include into her fingerprinting probe such that (1) the sensitivity against powerful attackers knowing the fingerprint distribution of the protected users (i.e., the worst-case distribution for the verifier) is bounded and the bound is set by the verifier, and (2) the usability cost[5] of collecting, storing, and using these attributes is close to being minimal. FPSelect is parameterized with the sensitivity requirement, the number of submissions that the attacker is deemed able to execute, and a representative sample of the fingerprints of the users.

The problem could be solved by exploring exhaustively the space of the possible attribute sets, evaluating the sensitivity and the usability cost of each set. This is, however, infeasible as the number of attribute sets grows exponentially with the number of attributes[6]. Moreover, we show below that the problem of finding the optimal attribute set is NP-hard. To the best of our knowledge, this is the first work that allows verifiers to dimension their fingerprinting probe in a sound manner, by quantifying the security level to reach, and selecting an attribute set that satisfies this level at a low usability cost.

Our key contributions are the following:

- We formalize the *attribute selection problem* that a verifier has to solve to dimension her probe. We show that this problem is NP-hard because it is a generalization of the Knapsack Problem. We define the model of the dictionary attacker, whose adversarial power depends on the knowledge of a fingerprint distribution. We propose a measure to quantify the sensitivity of a probe given a browser population and the number of fingerprints that the attacker is able to submit. We propose a measure of the usability cost that combines the size of the generated fingerprints, their collection time, and their instability.
- We propose a heuristic algorithm for selecting an attribute set that satisfies a higher bound on the sensitivity and reduces the usability cost. We express this as a search problem in the lattice of the power set of the candidate attributes. This algorithm is inspired by the Beam Search algorithm [25] and is part of the Forward Selection algorithms [50].
- We evaluate the FPSelect framework on a real-life fingerprint dataset, and compare it with common attribute selection methods based on the entropy and the conditional entropy. We show experimentally that FPSelect finds attribute sets that have a lower usability cost. The attribute sets found by FPSelect generate fingerprints that are 12 to 1,663 times smaller, 9 to 32,330 times faster to collect, and with 4 to 30 times less changing attributes between two observations, compared to the candidate attributes and on average. Compared to the baselines, the attribute sets found by FPSelect generate fingerprints that are up to 97 times smaller, are collected up to 3,361 times faster, and with up to 7.2 times less changing attributes between two observations, on average.

The rest of the paper is organized as follows. Section 2 defines the attack model and the attribute selection problem. Section 3 describes the resolution algorithm and the proposed illustrative measures of sensitivity and usability cost. Section 4 provides the results obtained by processing our framework and the baselines on a real-life fingerprint dataset. Section 5 discusses concrete usage of the framework. Section 6 describes the works related to the attribute selection problem. Finally, Section 7 concludes.

2 PROBLEM STATEMENT

In this section, we first present the considered authentication mechanism that relies on browser fingerprinting. Then, we describe how we model the attacker given his knowledge and possible actions. Finally, we pose the attribute selection problem that we seek to solve, and provide an example to illustrate the problem.

2.1 Authentication Mechanism

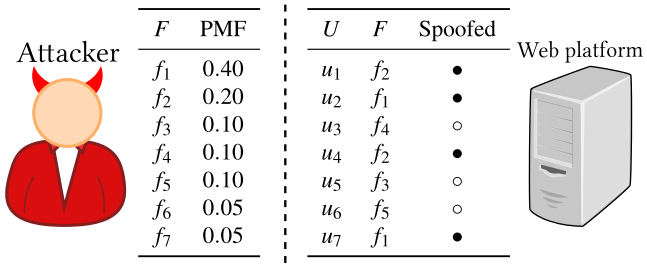
We consider the architecture and the three participants depicted in Figure 1. The authentication mechanism is executed on a *trusted*

[3] Most attributes are properties accessed through the browser that are limited by its functionalities. Other attributes are items which presence are checked (e.g., the fonts [18], the extensions [51]), or the computation of specific instructions (e.g., the HTML5 canvas [12]). These are limited by the available items or instructions, which can be large (e.g., more than 2^{154} for the canvas [31], nearly 30 thousand detectable extensions [27]).

[4] We emphasize that the candidate attributes can contain dynamic attributes, which can be used to implement challenge-response mechanisms that resist fingerprint replay attacks [31, 46]. We study nine instances of three dynamic attributes, which are the HTML5 canvas [12], the WebGL canvas [38], and audio fingerprinting methods [45].

[5] Any usability cost can be plugged (e.g, the privacy cost of including an attribute) provided that it is monotonic.

[6] Obviously, this discards as well the manual selection of attributes.



F	PMF
f_1	0.40
f_2	0.20
f_3	0.10
f_4	0.10
f_5	0.10
f_6	0.05
f_7	0.05

U	F	Spoofed
u_1	f_2	•
u_2	f_1	•
u_3	f_4	○
u_4	f_2	•
u_5	f_3	○
u_6	f_5	○
u_7	f_1	•

Figure 2: Example of an attacker instantiated with his knowledge of a probability mass function (PMF) over the fingerprints F, and a web platform protecting a user population U with their fingerprint. We consider a limit of two submissions and a strict comparison between the fingerprints. The attack dictionary is composed of f_1 and f_2, resulting in the shown impersonated users.

web platform and aims at authenticating *legitimate users* based on various authentication factors, including their browser fingerprint (in addition to, e.g., a password). For the sake of precision, we focus on the browser fingerprint and ignore the other factors.

A user is enrolled by providing his browser fingerprint to the verifier who stores it. During the authentication of a user, the fingerprint of the browser in use is collected by the fingerprinting probe of the verifier, and is compared with the fingerprint stored for the claimed account. If the collected fingerprint matches with the one stored, the user is given access to the account, and the stored fingerprint is updated to the newly collected one. The comparison is done using a *matching function* (i.e., a similarity function between two fingerprints that authorizes differences), as fingerprints are known to evolve [4, 13, 59]. Any matching function can be used provided that it is monotonic (i.e., if two fingerprints match[7] for an attribute set C, they also match for any subset of C). We explain in Section 2.3 the need for the monotonicity requirement, and refer to Section 4.2.4 for an example of a matching function. We consider one browser per user and discuss the extension to multiple browsers in Section 5.1.

2.2 Attack Model

The high-level goal of the attacker is to impersonate legitimate users in a *limited number of submissions* with the help of his knowledge, by forging a *fingerprint attack dictionary* similarly to dictionary attacks on passwords [10]. Figure 2 illustrates the attack that we consider. It shows an attacker with his knowledge of a fingerprint distribution, a population of protected users with their fingerprint, and the impersonated users. We define the attacker model in terms of background knowledge and possible actions, which are provided below and described further in the following subsections.

(1) The attacker cannot tamper with the web platform.
(2) The attacker cannot tamper with, nor eavesdrop, the communication between the users and the web platform.
(3) The attacker knows the attributes of the probe.
(4) The attacker knows a fingerprint distribution.
(5) The attacker can submit a limited number of arbitrary fingerprints.

[7]We stress that the monotonic property does not depend on the attributes.

2.2.1 Background Knowledge. The attacker can retrieve the attributes of the fingerprinting probe (assumption 3) by reverse-engineering the probe (e.g., static or dynamic analysis of the probe [6], analysis of the network packets).

The attacker knows the *domain of the fingerprints*, and can infer a fingerprint distribution (assumption 4) from documentation [47], datasets[8], or statistics[9] available online. He can also leverage phishing attacks [57], a pool of controlled browsers [41], or stolen fingerprints [34]. The weakest attacker is the one that lacks knowledge, and considers that the values of the attributes and fingerprints are uniformly distributed. His strategy is then to cover a space as large as possible of the fingerprint possibilities in the number of submissions authorized by the verifier. The strongest attacker is the one that manages to infer the exact fingerprint distribution among the users protected by the verifier. Additionally, our work can be easily extended to the attackers that partially know the fingerprints of targeted users[10] (see Section 5.3).

2.2.2 Actions. Tools exist for controlling the attributes[11] that compose the fingerprint (assumption 5), like Blink [32] or Disguised Chromium Browser [7]. Commercial solutions also exist, like AntiDetect[12] or Multilogin[13]. An attacker can also automatically alter the network packet that contains the fingerprint using tools like BurpSuite[14]. As these attacks are online guessing attacks [10, 60], we assume that the attacker is limited to a number of submissions per user.

2.2.3 Attacker Instance. The verifier instantiates an attacker by his knowledge of a fingerprint distribution, and by the number of submissions to which he is limited, to measure his reach.

2.3 Attribute Selection Problem

The defense problem consists into selecting the attribute set that composes the fingerprinting probe, to resist against an instantiated attacker and minimize the usability cost. On the one hand, including an attribute can reduce the reach of an attacker – called the *sensitivity* and measured as the *proportion of impersonated users* – because it adds one more information to distinguish different browsers. On the other hand, it increases the *usability cost* of the mechanism. For example, the fingerprints take more space to store, can take more time to collect, and can be more difficult to recognize due to the potentially induced instability.

The *Attribute Selection Problem* consists in finding the attribute set that provides the lowest usability cost and keeps the sensitivity below a threshold α set by the verifier[15]. Let A denote the set

[8]https://www.henning-tillmann.de/en/2014/05/browser-fingerprinting-93-of-all-user-configurations-are-unique
[9]http://carat.cs.helsinki.fi/statistics
[10]We do not consider the attackers that exactly know the fingerprint of the users they target (or their local configuration) because they are able to bypass trivially any fingerprinting authentication mechanism.
[11]These tools are able to control both the fixed and the dynamic attributes.
[12]https://antidetect.org
[13]https://multilogin.com
[14]https://portswigger.net/burp
[15]The *sensitivity threshold* α is defined by the verifier according to her security requirements. These requirements depend on the type of website that is to protect (e.g.,

User	CookieEnabled	Language	Timezone	Screen
u_1	True	fr	-1	1080
u_2	True	en	-1	1920
u_3	True	it	1	1080
u_4	True	sp	0	1920
u_5	True	en	-1	1080
u_6	True	fr	-1	1920

Table 1: Example of fingerprints shared by users.

of the candidate attributes. We consider an attribute set $C \subseteq A$, its usability cost $\mathrm{c}(C)$, and its sensitivity $\mathrm{s}(C)$. Any measure of usability cost and sensitivity can be plugged in FPSelect provided that it is *monotonic*. Indeed, the usability cost is required to be *strictly increasing* as we add attributes to an attribute set (e.g., the additional attributes are stored, which increases the storage cost). The sensitivity is required to be *monotonically decreasing* as we add attributes to an attribute set[16]. Indeed, adding an attribute to an attribute set should not higher the sensitivity because the added attribute either adds distinguishing information to the fingerprints or adds no information if it is strongly correlated with another attribute. For illustrative purposes, we propose measures of sensitivity and usability cost in Section 3. The ASP is thus formalized as searching for $\arg\min_{C \subseteq A}\{\mathrm{c}(C) : \mathrm{s}(C) \leq \alpha\}$.

2.4 Illustration of the Attribute Selection Problem

To illustrate the problem, we propose an example of a fingerprint distribution in Table 1. We consider an attacker who managed to infer the exact same distribution, and who is able to submit one fingerprint per user. If we solely include the `CookieEnabled` attribute which provides no distinctiveness, this attacker can impersonate every user by submitting the `True` value. Whereas including the `Language` and `Screen` attributes leads to unique fingerprints, which reduces the sensitivity to a sixth. Ignoring the `CookieEnabled` attribute reduces the usability cost without increasing the sensitivity. There is also an example of correlation. The `Timezone` and the `Language` attributes are the two most distinctive attributes, but including both does not improve the distinctiveness compared to considering `Language` alone.

3 ATTRIBUTE SELECTION FRAMEWORK

This section is dedicated to the description of our attribute selection framework. First, we show that the Attribute Selection Problem (ASP) is NP-hard because it is a generalization of the Knapsack Problem (KP), and remark that the ASP can be seen as a lattice of partial KP. Second, and consequently, we propose a greedy heuristic algorithm for finding solutions to the problem. Finally, we propose illustrative measures for the sensitivity and the usability cost.

a bank, a forum) and the contribution of browser fingerprints (e.g., the only secondary authentication factor, an additional verification among others [52]).

[16] The monotonicity requirement of the matching function comes from the monotonicity requirement of the sensitivity. Indeed, if the matching function was not monotonic, adding an attribute could result in a loss of distinctiveness (i.e., it is harder for the matching function to distinguish two browsers) and consequently in an increase of the sensitivity.

3.1 Similarity to the Knapsack Problem

The *Knapsack Problem* (KP) [28] is a NP-hard problem that consists into fitting valued-items into a limited-size bag to maximize the total value. More precisely, given a bag of capacity W and n items with their value v_i and their weight w_i, we search for the item set that maximizes the total value and which total weight does not exceed W. In this section, we show that the ASP is a generalization of the KP, therefore the ASP is NP-hard. We also provide a way to model the ASP as a lattice of partial KP.

First, we remark that the ASP can be solved by picking attributes until we reach the sensitivity threshold, or by starting from the candidate attributes and removing attributes successively without exceeding the threshold. We consider the latter and start from the set A of the candidate attributes. The *value* of an attribute set C is the cost reduction compared to the candidate attributes, formalized as $v(C) = \mathrm{c}(A) - \mathrm{c}(C)$. The *value* of an attribute a is the cost reduction obtained when removing a from C, which is formalized as $v(a|C) = \mathrm{c}(C) - \mathrm{c}(C \setminus \{a\})$. The *weight* of an attribute set C is its sensitivity with $w(C) = \mathrm{s}(C)$. The *weight* of an attribute a is the additional sensitivity induced by the attribute removal, formalized as $w(a|C) = \mathrm{s}(C \setminus \{a\}) - \mathrm{s}(C)$. The *capacity* W is the maximum sensitivity allowed, hence $W = \alpha$. As we remove attributes, the value increases (i.e., the usability cost decreases), and the weight (i.e., the sensitivity) may increase.

Theorem 3.1. *The Attribute Selection Problem is NP-hard.*

Proof. We consider a simple case where the attributes are not correlated. The weight and the value of the attribute a_i does not depend on the attributes already included in the probe, and is simply defined as w_i and v_i. We obtain a Knapsack Problem consisting into picking the attributes to remove from A, to maximize the total value and keep the weight under the threshold W. The ASP is therefore a generalization of the KP with relative weights and costs, making it at least as hard as the KP which is NP-hard. The Attribute Selection Problem is therefore NP-hard. □

3.1.1 The Attribute Selection Problem as a Lattice of Partial Knapsack Problems. The Attribute Selection Problem can be modeled as a lattice of partial Knapsack Problems (KP). We consider the deletive way that starts from the set A of the candidate attributes and removes attributes without exceeding the threshold. The initial partial KP consists into picking attributes from A to increase the value and keep the weight under W. The value and weight of each attribute $a \in A$ is $v(a|A)$ and $w(a|A)$. Once we pick an attribute a_p, a new partial KP arises: the item set is $A \setminus \{a_p\}$, the capacity is $W - w(a_p|A)$, and the value and weight of each attribute $a \in A \setminus \{a_p\}$ is now $v(a|A \setminus \{a_p\})$ and $w(a|A \setminus \{a_p\})$. Recursively, it holds for any set R of attributes to remove. The item set is then $A \setminus R$, the capacity is $W - w(R)$, and the value and weight of each attribute $a \in A \setminus R$ are $v(a|A \setminus R)$ and $w(a|A \setminus R)$. Following this, we are given a lattice[17] of partial KP to solve recursively, each node being a partial solution R, until we reach unfeasible problems (i.e., empty set of items, no more item can fit) and find a final solution among the partial solutions that reach this limit.

[17] This can be seen as a tree, but some paths lead to the same node. Indeed, removing the attributes a_1 then a_2 from A leads to the same partial problem as removing a_2 then a_1.

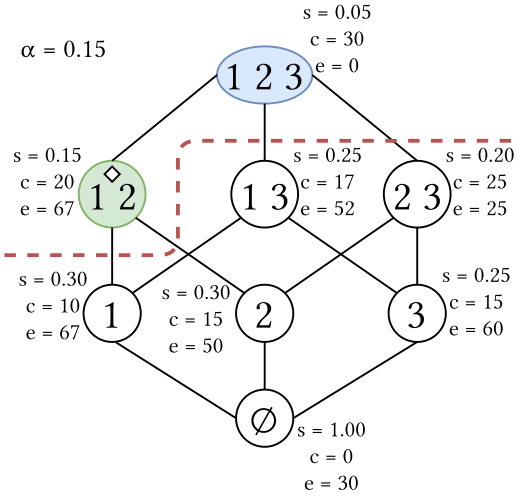


Figure 3: Example of a lattice of attribute sets, with their cost c, their sensitivity s, and their efficiency e. The blue node satisfies the sensitivity, the white nodes do not, and the green node with a diamond satisfies the sensitivity and minimizes the cost. The red line is the satisfiability frontier.

3.2 Lattice Model and Resolution Algorithm

In this section, we present how we model the possibility space as a lattice of attribute sets, and describe the greedy heuristic algorithm to approximately solve the ASP.

3.2.1 Lattice Model. The elements of the lattice are the *subsets* of A (A included) and the order is the *subset relationship* so that $C_i \prec C_j$ if, and only if, $C_i \subset C_j$. The efficiency of an attribute set C is the ratio between its cost reduction (i.e., $c(A) - c(C)$) and its sensitivity. Figure 3 shows an example of such lattice. The *satisfiability frontier* represents the transition between the attribute sets that satisfy the sensitivity threshold, and those that do not. The attribute sets just above this frontier satisfy the sensitivity threshold at a lower cost than any of their supersets. They comprise the solution found by our resolution algorithm and the optimal solution to the problem.

The sensitivity and the cost are bounded. The lower bound is located at the empty set, which has a sensitivity of 1.0 and a null usability cost. It is equivalent to not using browser fingerprinting at all. On the other end, the set composed of the candidate attributes A is a superset of every attribute set, and provides the lowest sensitivity and the highest usability cost. If A does not satisfy the sensitivity threshold, there is no solution as any other subset has a higher or equal sensitivity.

3.2.2 Greedy Algorithm. We propose the greedy heuristic algorithm presented in Algorithm 1 to find good solutions to the Attribute Selection Problem. It consists into a bottom-up exploration of the lattice by following k paths until reaching the satisfiability frontier. The higher k is, the larger is the explored space, but the higher is the computing time. This algorithm is inspired by the model of the ASP as a lattice of partial Knapsack Problems, and by the Beam Search algorithm [25]. The similarity with the latter lies

Data: The candidate attributes A, the sensitivity threshold α, the number of explored paths k.
Result: The attribute set of the explored paths that satisfies the sensitivity threshold at the lowest cost.

$c_{min}, T, I \leftarrow \inf, \emptyset, \emptyset$
$S \leftarrow$ a collection of k empty sets
if $s(A) > \alpha$ **then**
  Quit as no solution exists
end
while *S is not empty* **do**
  $E \leftarrow \{C = S_i \cup \{a\} : \forall S_i \in S, \forall a \in A \setminus S_i, \nexists C' \in T \cup I, C' \subset C\}$
  $S \leftarrow \emptyset$
  for $C \in E$ **do**
    if $s(C) \leq \alpha$ **then**
      $T \leftarrow T \cup \{C\}$
      $c_{min} \leftarrow c(C)$ if $c(C) < c_{min}$
    end
    else if $c(C) < c_{min}$ **then**
      $S \leftarrow S \cup \{C\}$
    end
    else
      $I \leftarrow I \cup \{C\}$
    end
  end
  $S \leftarrow$ the k most efficient attribute sets C of S according to $\frac{c(A)-c(C)}{s(C)}$
end
return $\arg\min_{C \in T} c(C)$

Algorithm 1: Greedy algorithm to find good solutions to the Attribute Selection Problem.

in the successive expansion of a limited number of nodes, that are chosen according to an order between partial solutions. The order is the efficiency in our case. Our proposed algorithm is part of the Forward Selection algorithms [50], as it iteratively picks attributes according to a criterion, and takes into account those already chosen. However, the proposed algorithm provides the ability to explore several sub-solutions instead of a single one, includes pruning methods that help reduce the computing time, and stops when it reaches the satisfiability frontier instead of when the criterion is not statistically improved.

Algorithm Working. Algorithm 1 works by exploring k paths of the lattice. It starts from the empty set and stops when every path reaches the satisfiability frontier. The collection S holds the attribute sets to expand and is initialized to k empty sets. At each stage, the attribute sets to explore are stored in the collection E. They consist of each $S_i \in S$ with one additional attribute. The cost and the sensitivity of each attribute set $C \in E$ is then measured. If C satisfies the sensitivity threshold α, it is added to the collection T of the attribute sets that reach the satisfiability frontier, otherwise it is added to the collection S of the attribute sets to expand. Finally, the collection S is updated to only hold the k most efficient attribute sets. The efficiency of an attribute set is the ratio between its gain (i.e., the cost reduction compared to the candidate attributes) and its sensitivity. All this process is repeated until S is

Stage	E	T	S
1	$\{\{1\},\{2\},\{3\}\}$	$\{\}$	$\{\{1\},\{3\}\}$
2	$\{\{1,2\},\{1,3\},\{2,3\}\}$	$\{\{1,2\}\}$	$\{\{1,3\}\}$
3	$\{\}$	$\{\{1,2\}\}$	$\{\}$

Table 2: Example of the execution of Algorithm 1 on the lattice of Figure 3, with the sensitivity threshold $\alpha = 0.15$ and the number of explored paths $k = 2$. Stage i is the state at the end of the i-th `while` loop.

empty, when all the k paths have reached the satisfiability frontier. The solution is then the attribute set of the lowest cost in T.

Pruning Methods. Three properties allow us to reduce the number of attribute sets that are explored. First, we hold the minimum cost c_{min} of the attribute sets T that satisfy the sensitivity. Any explored attribute set that has a cost higher than c_{min} is not added to the collection S of those to explore. Indeed, this attribute set does not provide the lowest cost, nor do its supersets. Then, during the expansion of two attribute sets S_i and S_j of the same size, if S_i satisfies the sensitivity and S_j does not, they can have a common superset S_l. In this case, S_l does not need to be explored as it costs more than S_i. We store S_i in I so that we can check if an attribute set C has a subset in I, in which case we do not explore C. The same holds if S_i costs less than c_{min} and S_j costs more than c_{min}.

Algorithm Complexity. Starting from the empty set, we have n supersets composed of one more attribute. From these n supersets, we update S to hold at most k attribute sets. The attribute sets $S_i \in S$ are now composed of a single attribute, and each S_i has $n-1$ supersets composed of one additional attribute. At any stage, we have at most kn attribute sets to explore. This process is repeated at most n times, as we can add at most n attributes, hence the *computational complexity* of the Algorithm 1 is of $O(kn^2\omega)$, with ω being the computational complexity of the measures of usability cost and sensitivity of an attribute set. The collection E contains at most kn attribute sets (at most n supersets for each $S_i \in S$). The collections S, T, and I can contain more sets, but are bounded by the number of explored nodes which is kn^2. The *memory complexity* of the Algorithm 1 is then of $O(kn^2)$.

Example. Table 2 displays an example of the execution of Algorithm 1 on the lattice presented in Figure 3, with the sensitivity threshold $\alpha = 0.15$ and the number of explored paths $k = 2$. The stage i corresponds to the state at the end of the i-th `while` loop. Initially, the collection S is $\{\emptyset, \emptyset\}$. At stage 1, the two most efficient attribute sets of E are $\{1\}$ and $\{3\}$, which are stored into S. At stage 2, we assume that the attribute set $\{1,2\}$ is measured first as there is no order among E. In this case, this attribute set is added to the collection T, and the minimum cost is now 20. The attribute set $\{1,3\}$ is then added to S, but $\{2,3\}$ is not as it has a higher cost than the minimum cost. At stage 3, the attribute set $\{1,2,3\}$ is not added to the collection E as it is a superset of one attribute set of the collection T. The final solution is the less costly attribute set of T, which is $\{1,2\}$ in this case, and happens to be the optimal solution.

3.3 Illustrative Measures of Sensitivity and Usability Cost

In this section, we illustrate a sensitivity measure as the proportion of impersonated users given the strongest attacker of our model that knows the fingerprint distribution among the protected users. We also illustrate a usability cost measure according to the fingerprints generated by a fingerprinting probe on a browser population.

3.3.1 Sensitivity Measure. We measure the sensitivity of a given attribute set according to an instantiated attacker and a population of users sharing browser fingerprints. The attacker knows the fingerprint distribution of the protected users, and submits orderly the most probable fingerprints, until reaching the threshold on the number of submissions. The illustrative *sensitivity measure* evaluates the proportion of users that are impersonated considering the matching function.

From an attribute set C, we retrieve the fingerprint domain F_C such that $F_C = \prod_{a \in C} \text{domain}(a)$, with $\text{domain}(a)$ being the domain of the attribute a and $\prod$ being the Cartesian product. We denote F_A the fingerprints when considering the set A of the candidate attributes. We denote U the set of the users that are protected by the verifier. The set $\mathcal{M} = \{(u, f) : u \in U, f \in F_A\}$ represents the mapping from the users to their fingerprint, so that the user u has the fingerprint f stored.

We denote $\text{project}(f, C)$ the function that projects the fingerprint $f \in F_{C'}$ from the set of attributes C' to the set of attributes C, with the requirement that $C \subseteq C'$. Finally, the function denoted $\text{dictionary}(p, F_C, \beta)$ retrieves the β-most probable fingerprints of F_C given the probability mass function p. We note that it is trivial to retrieve the distribution of the fingerprints composed of any attribute subset $C \subset A$ from the distribution of the fingerprints composed of the candidate attributes A.

We denote $f[a]$ the value of the attribute a for the fingerprint f, and $f[a] \approx^a g[a]$ the matching between the value of the attribute a for the stored fingerprint f and the submitted fingerprint g. It is true only if $f[a]$ matches with $g[a]$, meaning that $g[a]$ is deemed a legitimate evolution of $f[a]$. Finally, we define the set of the matching functions of each candidate attribute as $\Phi = \{\approx^a : a \in A\}$.

We measure the sensitivity as the proportion of impersonated users among a population of protected users, against the attacker that knows the fingerprint distribution among them, using Algorithm 2. The illustrative sensitivity measure is *monotonic* as demonstrated in Appendix A.

The *number of submissions* is defined by the verifier according to her rate limiting policy [17] (e.g., blocking the account after three failed attempts). This limit could be set to 1 as a user cannot mistake his browser fingerprint. However, a user can browse from a new or a public browser, and taking preventive action on this sole motive is unreasonable.

3.3.2 Usability Cost Measure. There is no off-the-shelf measure of the usability cost of the attributes (e.g., the `UserAgent` HTTP header [47] has no specified size, collection time, nor change frequency). This cost also depends on the fingerprinted population (e.g., mobile browsers generally have fewer plugins than desktop browsers, resulting in smaller values for the list of plugins [5]). As

Data: The attribute set C, the limit on the number of submissions β, the mapping $\mathcal{M}$ from the users to their browser fingerprint, the probability mass function p, and the set Φ of matching functions.
Result: The proportion of impersonated users.
$R \leftarrow \{\}$
$F_C \leftarrow$ the fingerprint domain when considering C
$V \leftarrow \text{dictionary}(p, F_C, \beta)$
forall $(u, f^*) \in \mathcal{M}$ **do**
  $f \leftarrow \text{project}(f^*, C)$
  if $\exists g \in V$ *st.* $\forall a \in C, f[a] \approx^a g[a]$ **then**
    $R \leftarrow R \cup \{u\}$
  end
end
return $\frac{\text{card}(R)}{\text{card}(U)}$

Algorithm 2: Illustrative sensitivity measure.

a result, we design an illustrative cost measure that combines three sources of cost (i.e., space, time, and instability), which is computed by the verifier on her fingerprint dataset. The *fingerprint dataset* used to measure the costs is denoted $D = \{(b, f) : b \in B, f \in F_A\}$, with B being the set of observed browsers.

The *memory cost* is measured as the average fingerprint size. The attribute values are stored and not compressed into a single hash, which is necessary due to their evolution through time. We denote $\text{mem}(C, D)$ the *memory cost* of the attribute set C, and $\text{size}(x)$ the size of the value x. The memory cost is defined as

$$\text{mem}(C, D) = \frac{1}{\text{card}(D)} \sum_{(b,f) \in D} \sum_{a \in C} \text{size}(f[a]) \tag{1}$$

The *temporal cost* is measured as the average fingerprint collection time, and takes into account the asynchronous collection of some attributes. Although attributes can be collected asynchronously, some require a non-negligible collection time (e.g., the dynamic attributes [38, 45]). We denote $\text{time}(C, D)$ the temporal cost of the attribute set C. Let A_{seq} be the set of the sequential attributes, and A_{async} the set of the asynchronous attributes, so that we have $C = A_{\text{seq}} \cup A_{\text{async}}$. Let $\text{t}(b, f[a])$ be the collection time of the attribute a for the fingerprint f collected from the browser b. The temporal cost is defined as

$$\text{time}(C, D) = \frac{1}{\text{card}(D)} \sum_{(b,f) \in D} \max(\{\text{t}(b, f[a]) : a \in A_{\text{async}}\} \cup \{\sum_{s \in A_{\text{seq}}} \text{t}(b, f[s])\}) \tag{2}$$

The *instability cost* is measured as the average number of changing attributes between two consecutive observations of the fingerprint of a browser. We denote $\text{ins}(C, D)$ the instability cost of the attribute set C. We denote $\mathcal{C}(D)$ the non-empty set of the consecutive fingerprints coming from the same browser in the dataset D, and $\delta(x, y)$ the Kronecker delta being 1 if x equals y and 0 otherwise. The instability cost is defined as

$$\text{ins}(C, D) = \frac{1}{\text{card}(\mathcal{C}(D))} \sum_{(f,g) \in \mathcal{C}(D)} \sum_{a \in C} \delta(f[a], g[a]) \tag{3}$$

The three dimensions of the cost are weighted by a three-dimensional *weight vector* denoted $\gamma = [\gamma_1, \gamma_2, \gamma_3]$ such that the weights are strictly positive numbers. The verifier tunes these weights according to her needs (e.g., allowing fingerprints to be more unstable, but requiring a shorter collection time). She can do this by defining an equivalence between the three dimensions (e.g., one millisecond of collection time is worth ten kilobytes of size), and setting the weights so that these values amount to the same quantity in the total cost. For a concrete example, we refer to Section 4.2.3.

Finally, we denote $\text{cost}(C, D)$ the cost of the attribute set C given the fingerprint dataset D. The illustrative usability cost measure is *monotonic* as demonstrated in Appendix A, and is formalized as

$$\text{cost}(C, D) = \gamma \cdot [\text{mem}(C, D), \text{time}(C, D), \text{ins}(C, D)]^\intercal \tag{4}$$

4 EXPERIMENTAL VALIDATION

In this section, we describe the experiments that we perform to validate our framework. We begin by presenting the fingerprint dataset that is used, and describing how the usability cost and the matching function are implemented. Then, we present the results of the attribute selection framework executed with different parameters, and compare them with the results of the common baselines. The experiments were performed on a desktop computer with 32GB of RAM and 32 cores running at 2GHz.

4.1 Fingerprint Dataset

The fingerprint dataset used in this work is the same dataset as the one studied by Andriamilanto et al. in [4, 5]. It was collected from December 7, 2016, to June 7, 2017, during a real-life experiment in which the authors integrated a fingerprinting probe to two pages of one of the 15 most visited websites in France. We refer to their studies [4, 5] that provide an in-depth analysis of this dataset, a comprehensive description of the fingerprint collection, a precise description of the preprocessing steps that include a cookie resynchronization process similar to [13], and an exhaustive list of the attributes with their properties. In a nutshell, the preprocessed dataset contains 5, 714, 738 entries (comprising identical fingerprints for a given browser if interleaved[18]) and 4, 145, 408 fingerprints (no identical fingerprint counted for the same browser), that are collected from 1, 989, 366 browsers. The instability is evaluated from 3, 725, 373 pairs of consecutive fingerprints, coming from the 27.53% of browsers that have multiple entries. The fingerprints are composed of 253 candidate attributes, of which 49 attributes are completely correlated with another one [5], so that knowing the value of the other attribute allows to completely infer their value. Although these attributes are correlated, we cannot simply remove them as the attributes present dissimilar costs that also depend on the attributes that are already considered.

[18] A browser can present interleaved fingerprints [5] like a, b, then a again. They typically come from a switch between two environments, like a laptop to which an external screen is plugged and unplugged. This browser has 3 entries, but only has 2 fingerprints (a and b) to avoid over counting. These interleaved fingerprints are held when measuring the instability cost.

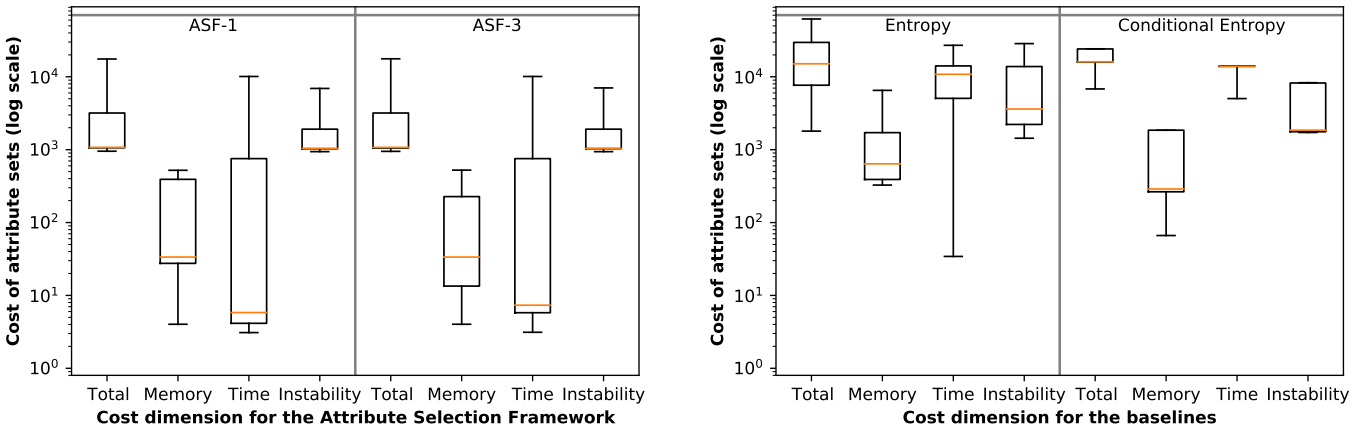


Figure 4: Cost of the attribute sets found by the ASF with 1 explored path (ASF-1), the ASF with 3 explored paths (ASF-3), the entropy, and the conditional entropy. The costs are in points, so that 10,000 additional points increases the size of fingerprints by 10 kilobytes, their collection time by 1 second, or the number of changing attributes between observations by 1 attribute, on average. A solution exists for 9 of the 12 cases. The gray horizontal line is the cost when considering all the candidate attributes.

4.2 Instantiation of the Experiments

In this section, we present the instantiation of the parameters for the experiments. We first describe how the verifier and the attacker are instantiated by presenting the chosen user population, sensitivity thresholds, and number of submissions. Then, we detail the implementation of the usability cost measure and the matching function between fingerprints, alongside the value of the parameters or weights that they use.

4.2.1 Verifier Instantiation. On the verifier side, we simulate a *user population* by randomly sampling 30,000 browsers from the first month of the experiment to represent a medium-sized website. The observed fingerprint is considered as the fingerprint stored for the user who owns the browser. We configure the resolution algorithm to have 1 and 3 *explored paths* to compare the gain achieved by a larger explored space. We call *ASF-1* and *ASF-3* our attribute selection method with respectively 1 and 3 explored paths. We consider the set of *sensitivity thresholds* $\{0.001, 0.005, 0.015, 0.025\}$. Bonneau et al. [11] defined the resistance against online attacks as a compromise of 1% of accounts after a year when 10 guesses per day are allowed. Hayashi et al [21] estimated that 0.001 is equivalent to a random guess of four-digit. However, to the best of our knowledge, no standard value exists. Hence, we make the choice of these values starting from 0.001 and going to 0.025 to obtain a range from a strict security requirement to one that is less strict. We admit that 0.025 (2.5%) is already high, but it is close to the proportion of users that share the 10 most common passwords in previously leaked datasets [60].

4.2.2 Attacker Instantiation. On the attacker side, an instance is parameterized with the number of fingerprints β that he can submit, and his knowledge over the fingerprint distribution. We consider the strongest attacker of our attack model that knows the fingerprint distribution among the user population.

We consider the set of number of submissions $\{1, 4, 16\}$. To the best of our knowledge, no standard value exists. The choice of 1 is for a strict rate limiting policy that blocks the account on any failure and asks the user to change his password. The choice of 4 is for a policy that would require a CAPTCHA after 3 failed attempts, and would perform the blocking and password change after the fourth failed attempt. Finally, the choice of 16 is for a policy that would let more attempts before performing the blocking and password change. The chosen values are close to the number of submissions allowed into policies enforced in real life [17], and the ones estimated as reasonable values against online guessing attacks [10].

Value	Cost (pts)	Memory (B)	Time (s)	Inst. (chgs)
Candidate	134,270	6,114	9.98	2.83
Max. cost	99,846	1,102	9.98	0.51
Avg. cost	8,794	26	0.87	$1.13\ 10^{-2}$
Min. cost	1	1	0	0.00

Table 3: The cost of the 253 candidate attributes, together with the maximum, the average, and the minimum cost of a single attribute for each cost dimension.

4.2.3 Implementation of the Usability Cost Measure. The implemented *usability cost function* measures the memory in bytes (a character per byte), the time in milliseconds, and the instability as the average number of changing attributes between the consecutive fingerprints. We configure the three-dimensional *weight vector* to the values $\gamma = [1; 10; 10,000]$ to have an equivalence between 10 kilobytes, 1 second, and 1 changing attribute on average, which are all equal to 10,000 points. Table 3 displays the cost of the 253 candidate attributes, together with the minimum, the average, and the maximum cost of a single attribute for each cost dimension.

4.2.4 Implementation of the Matching Function. The implemented *matching function* checks that the distance between the attribute

values of the submitted fingerprint and the stored fingerprint is below a threshold. Similarly to previous studies [13, 26, 59], we consider a distance measure that depends on the type of the attribute. The minimum edit distance [24] is used for the textual attributes, the Jaccard distance [62] is used for the set attributes, the absolute difference is used for the numerical attributes, and the reverse of the Kronecker delta (i.e., $1 - \delta(x, y)$) is used for the categorical attributes. The dynamic attributes (e.g., HTML5 canvas) are matched identically (i.e., using a threshold of 1) as they serve the challenge-response mechanism. More complex matching functions exist (e.g., based on rules and machine learning [59]). They can be integrated to the framework as long as they are monotonic[19].

The *distance threshold* for each attribute is set using Support Vector Machines (SVM) [22] and the following methodology. First, we split our dataset into 6 samples (one for each month) and extract the positive and negative classes. They respectively consist into the consecutive fingerprints of a browser, and two randomly picked fingerprints of different browsers. We assume that a user spends at most one month between each connection, and otherwise would accept to process a heavier fingerprint update process. Then, for each attribute, we train an SVM model on the two classes of each monthly sample, and extract the threshold from the resulting hyperplane. Finally, we compute the average of the 6 obtained thresholds to get the distance threshold for each attribute.

4.2.5 Baselines. We compare our method with common attribute selection methods. The entropy-based method [8, 23, 29, 35] consists into picking the attributes of the highest entropy until reaching an arbitrary number of attributes. The method based on the conditional entropy [14] consists into iteratively picking the most entropic attribute according to the attributes that are already chosen, and re-evaluating the conditional entropy of the remaining attributes at each step, until an arbitrary number of attributes is reached. Instead of limiting to a given number of attributes, we pick attributes until the obtained attribute set satisfies the sensitivity threshold. For simplification, we call *entropy* and *conditional entropy* the attribute selection methods that rely on these two metrics.

4.3 Attribute Selection Framework Results

In this section, we present the results obtained on the previously presented dataset by processing the Attribute Selection Framework on the instantiated attackers, and compare them with the results of the baselines. The results are obtained for the 12 cases consisting of the Cartesian product between the values of the sensitivity threshold α and those of the number of submissions β. We present here the obtained results and discuss the attributes that are the most selected by the framework. The exhaustive list of the selected attributes is provided in Appendix C.

4.3.1 Key Results. The attribute sets found by the Attribute Selection Framework (ASF) generate fingerprints that are 12 to 1,663 times smaller, 9 to 32,330 times faster to collect, and with 4 to 30 times less changing attributes between two observations, compared to the candidate attributes and on average. Compared to the attribute sets found by the baselines, the ones found by the ASF-1 generate fingerprints that are up to 97 times smaller, are collected up to 3,361 times faster, and with up to 7.2 times less changing attributes between two observations, on average. These gains come with a higher computation cost, as the ASF-1 explores more attribute sets by three orders of magnitude compared to the baselines. However, the implemented attribute sets can be updated rarely, and the usability gain is reflected on each authentication performed by each user.

Increasing the number of explored paths by the ASF to three does not significantly change the results. The attribute sets found by the ASF-3 can have a lower usability cost, or a higher usability cost due to local optimum (see Section 4.3.4). We show that even when considering all of our candidate attributes, the strongest attacker that is able to submit 4 fingerprints can impersonate 63 users out of the 30,000 users of our sample. If this attacker is able to submit 16 fingerprints, this number increases to 152 users.

4.3.2 Results of the Attribute Selection Framework. Figure 4 displays the cost of the attribute sets found by the ASF with 1 explored path (ASF-1), the ASF with 3 explored paths (ASF-3), the entropy, and the conditional entropy. The costs are in points, so that 10,000 additional points increase the size of fingerprints by 10 kilobytes, their collection time by 1 second, or the number of changing attributes between observations by 1 attribute, on average. There is a solution for 9 out of the 12 cases. The cases without a solution are discussed in Section 4.3.6.

Half of the attribute sets found using the ASF-1 generate fingerprints that, on average, have a size lower than 34 bytes (less than 522 bytes for all sets), are collected in less than 0.59ms (less than 1.01 seconds for all sets), and have less than 0.02 changing attributes between two observations (less than 0.07 attributes for all sets). Compared to the candidate attributes and on average, the generated fingerprints are 12 to 1,663 times smaller, 9 to 32,330 times faster to collect, and with 4 to 30 times less changing attributes between two observations.

The difference in usability cost of the attribute sets found by the ASF-3 and the ASF-1 is negligible. The attribute sets found by the ASF-3 are as less costly as they are more costly than the attribute sets found by the ASF-1. This results in the median additional cost of each dimension being zero. The average resulting fingerprint is from 198 bytes smaller to 30 bytes larger, takes from 3ms less to collect to 0.3ms more, and has from 0.03 less changing attributes to 0.04 more. Exploring more paths can counter-intuitively provide a higher usability cost, due to the local optimum problem described in Section 4.3.4. Indeed, when exploring more nodes, the followed paths can diverge as we hold more temporary solutions, which can be local optimum. The computation cost of increasing the number of explored paths is not worth the expected gain in our experimental setup.

4.3.3 Comparison with the Baselines. The ASF-1 finds attribute sets that consume less resources than the baselines in all the 9 cases having a solution. The attribute sets found by the *entropy* consume more resources than the attribute sets found by the ASF-1, with a total cost from 1.8 to 14 times higher. The average generated fingerprint by the attribute sets chosen by the entropy, compared to the attribute sets chosen by the ASF-1, is from 1.6 to 97 times larger,

[19] A matching function is monotonic if two fingerprints that match for an attribute set C also match for any subset of C.

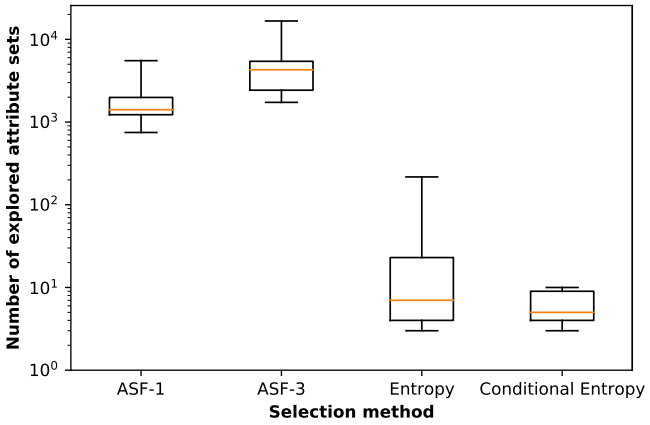


Figure 5: Number of explored attribute sets by the attribute selection methods.

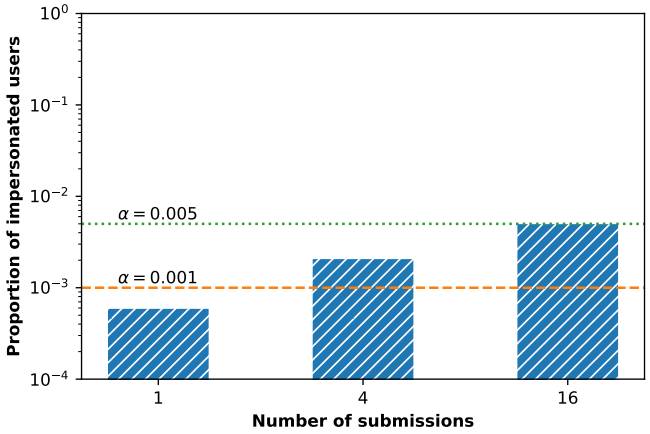


Figure 6: Proportion of impersonated users among the 30,000 users, considering the candidate attributes, the matching function, and as a function of the number of submissions. The sensitivity thresholds α that have no solution for some cases are displayed.

has a collection time that is from 1.5 to 1,872 times higher, and has from 1.5 to 7.2 times more changing attributes between the consecutive fingerprints. The attribute sets found by the *conditional entropy* consume more resources than the attribute sets found by the ASF-1, with a total cost from 1.3 to 15 times higher. The average generated fingerprint by the attribute sets chosen by the conditional entropy, compared to the attribute sets chosen by the ASF-1, is from 1.5 to 16 times larger, has a collection time that is from 1.3 to 3,361 times higher, and has from 1.1 to 4.7 times more changing attributes between the consecutive fingerprints.

4.3.4 Reasons for Sub-optimal Results. The sub-optimal solutions that are found by the Attribute Selection Framework are due to a problem of local optimum. At a given step, the most efficient attribute sets S can have supersets of higher cost than the supersets of another S'. The supersets of S are then explored, whereas the less costly supersets here would have been the supersets of S'.

4.3.5 Computation Cost. The attribute selection framework has a higher computation cost. Indeed, at each stage of the exploration, the ASF explores up to $n - 1$ attribute sets[20] for each temporary solution, with n being the number of candidate attributes. The ASF-1 explores more attribute sets by three orders of magnitude compared to the baselines. However, this is an upper bound as the baselines require preprocessing. Indeed, the attributes has to be sorted by their entropy or by their conditional entropy.

Figure 5 displays the number of attribute sets explored by the attribute selection methods. The number of explored attribute sets by the ASF-1 goes from 748 to 5,522. The ASF-3 explores approximately 3 times more attribute sets than the ASF-1: from 1,730 to 16,659 explored attribute sets. The number of attribute sets explored by the entropy ranges from 3 to 217, and from 3 to 10 for the conditional entropy. However, the conditional entropy method requires to sort the n attributes by their conditional entropy, which requires $\sum_{i=0}^{n} n - i$ steps. This difference of explored attribute sets between the entropy and the conditional entropy is explained by the latter avoiding selecting correlated attributes.

4.3.6 Lower Bound on the Impersonated Users. The obtained sensitivity against our instantiated attackers ranges from the minimum sensitivity when considering the candidate attributes, as displayed in Figure 6, to the maximum sensitivity of 1.0 when considering no attribute at all. All the possible attribute sets have their sensitivity comprised between these two extremum. The instantiated attackers that are allowed 4 submissions are able to impersonate 63 users out of the 30,000 users, which exceeds the sensitivity threshold of 0.001. When allowed 16 submissions, the number of impersonated users increases to 152, which exceeds the sensitivity threshold of 0.005 that corresponds to 150 users.

4.3.7 Selected Attributes. We have 9 combinations of sensitivity threshold and number of submissions that show a solution. The attribute selection framework is executed twice with two number of explored paths (1 and 3), hence we have 18 cases for which it found a solution. We discuss below the six most selected attributes that are selected in more than five cases. We remark that they concern hardware and software components that we expect to not be correlated. Indeed, we do not expect a strong link to exist between the browser window size, the number of logical processor cores, the graphics driver, the browser version, a scheme drawn in the browser, and the type of network connection. The results about the attributes that we present here come from their analysis by Andriamilanto et al. [5].

The `innerHeight` property of the `window` JavaScript object provides the height of the visible part of the browser window. It is selected in all the cases, and is the first attribute to be selected during the exploration as it provides the highest efficiency. Indeed, its usability cost is low with, on average, a size of 3.02 bytes, a

[20] The set of the explored attribute sets can overlap as two temporary solutions can have a common superset.

collection time of 0.14ms, and a change between 9.38% of the observations. Moreover, it is the fifth most distinctive attribute of our dataset, with an entropy of 8.53 bits and the most common value being shared by 2.70% of the fingerprints.

The `hardwareConcurrency` property that is collected from the `navigator` JavaScript object provides the number of logical processor cores of the device that runs the browser. This attribute is selected in 11 cases, and shows a high efficiency mostly due to the very low usability cost. Indeed, it shows, on average, a size of 1 byte (the value is majoritarily a single digit), a collection time of 0.17ms, and a change between 0.11% of the observations. However, it shows a lower distinctiveness, with an entropy of 1.88 bits and the most common value being shared by 39.64% of the fingerprints.

The `UNMASKED_RENDERER_WEBGL` property of an initialized WebGL Context provides a textual description of the graphics driver. This attribute is selected in 8 cases and shows a high efficiency. Indeed, it has, on average, a size of 24.51 bytes, a collection time of 0.27ms, and a change between 0.91% of the observations. Although the most common value is shared by 28.27% of the fingerprints, it still provides an entropy of 5.89 bits.

The `appVersion` property of the `navigator` JavaScript object provides the version of the browser. This attribute is selected in 7 cases. It shows, on average, a size of 101.76 bytes, a collection time of 0.13ms, and a change between 1.57% of the observations. Although the most common value is shared by 22.61% of the fingerprints, it still provides an entropy of 7.52 bits.

The HTML5 canvas inspired by the AmIUnique study [33] is selected in 7 cases, mainly due to its high distinctiveness. It has, on average, a size of 63.98 bytes, a collection time of 71.17ms, and a change between 1.36% of the observations. It shows an entropy of 7.76 bits, and the most common value is shared by 7.09% of the fingerprints.

The `connection.type` property of the `navigator` JavaScript object provides the type of the network connection in use by the browser. This attribute is selected in 6 cases, mainly due to its low usability cost. It provides, on average, a size of 1.47 bytes, a collection time of 0.21ms, and a change between 0.80% of the observations. It shows a lower distinctiveness compared to the other attributes, with an entropy of 0.61 bits and the most common value being shared by 89.52% of the fingerprints.

5 DISCUSSION

5.1 Usage of Multiple Browsers

Users tend to browse websites using multiple devices, typically a desktop and a mobile device[21]. FPSelect can be extended to support the usage of multiple browsers by the users, by changing the sensitivity measure so that a user is impersonated if one of his browsers is spoofed by the attacker. Using a monotonic matching function (e.g., the matching function described in Section 4.2.4), this sensitivity measure is also monotonic[22]. Boda et al. [9] showed that some attributes provide information about the underlying system (e.g., the list of fonts) and can be used for cross-browser fingerprinting. Although such technique is interesting in an authentication context (e.g., recognizing the common attributes between the browsers of a user), this is out of the scope of our work.

5.2 Update of Attributes through Time

The verifier can keep the attributes of the fingerprinting probe up to date by re-executing the framework. To do so, she performs a fingerprint collection on a browser population close to the population of her web platform, using a wide-surface of fingerprinting attributes. We emphasize that the usability requirement is less strict for such experiment (e.g, the fingerprints can take more time or more space). Web technologies do not evolve frequently. For example, Andriamilanto et al. [5] analyze the dataset that we study, and do not observe any significant change over the 6 months of the experiment. Moreover, changing the attribute set requires to update the fingerprint that is stored for each user. Hence, the verifier can – and should – perform this process rarely (e.g., once per semester or per year). Finally, the verifier can monitor the distinctiveness and the stability of the stored fingerprints, and perform an update if a drastic change is detected (e.g., an attribute becomes highly unstable [30] or homogeneous).

5.3 Attribute Sets in a Per-Browser Basis

The attribute set can also be selected in a per-browser basis, but FPSelect is not designed for this. However, it is possible to execute FPSelect on subpopulations of browsers (e.g., mobile and desktop browsers) to obtain an attribute set per subpopulation. To do so, the whole framework is simply executed on the subpopulation of browsers. The sensitivity measure then considers that the attacker focus on this subpopulation (i.e., he knows the fingerprint distribution of this subpopulation). The costs are also specifically measured on the subpopulation (e.g., the list of plugins is most of the time empty for the mobile browsers [5, 53], hence is less costly for this subpopulation). The thresholds of the subsets have to be set so that the overall sensitivity threshold is satisfied. The simplest way is to set the thresholds of the subsets to the overall threshold. Indeed, if less than x percent of the users of each subset are impersonated, less than x percent of the overall users are.

6 RELATED WORKS

6.1 Attribute Selection

Previous works identify the need to reduce the included attributes, and used various methods to perform the selection. Most of the previous works either remove the attributes of the lowest entropy [59], or leverage greedy algorithms that iteratively pick the attributes of the highest weight (typically the entropy) until a threshold (typically on the number of attributes) is reached [8, 23, 29, 35, 56]. These methods do not consider the correlation that can occur between the attributes. Indeed, two attributes can separately have a high entropy, but, when taken together, provide a lower entropy than another attribute set. Table 1 displays a concrete example of such case.

To the best of our knowledge, two works take the correlation into account in their attribute selection method. Fifield and Egelman [14] weigh the attributes by the conditional entropy given

[21] https://www.javelinstrategy.com/coverage-area/how-online-vs-mobile-shifting-browser-vs-app

[22] Indeed, if the fingerprint of one of the user's browsers matches with the fingerprint of the attacker for an attribute set C, it also matches for any subset of C.

the attributes that are already picked. The conditional entropy of each attribute is updated on each turn, and picking two correlated attributes is therefore avoided. Pugliese et al. [44] propose two attribute selection methods that iteratively pick the attribute that maximizes a criterion, given the attributes that are already chosen. The first criterion to maximize is the number of users for which their fingerprints are not shared by any other user and stay identical between at least two observations. The second criterion to maximize is the duration for which these fingerprints stay identical. These two works only maximize one criterion, and ignore the usability cost of the attributes. On the contrary, our framework performs a trade-off between the sensitivity that is tied to the distinctiveness, and the usability cost that has a stability dimension.

Gulyás et al. [18] study the problem of finding the set of s-items (e.g., fonts, plugins, applications) for which to check the presence on a device, to reduce the number of devices that agree on the same value. They prove that this problem is NP-hard, and propose greedy algorithms to find the closest approximation in polynomial time. Our problem is different because we do not choose the items to check the presence for, that consist of binary value, but on selecting the categorical attributes to collect. Moreover, they seek to reduce the number of collected attributes, and to minimize the users that agree on the same values. We seek to reduce the sensitivity against dictionary attackers, and to minimize the usability cost that comprises various aspects. Indeed, the attributes are not equal regarding their usability cost (e.g., some are collected almost instantly whereas others take seconds).

Flood and Karlsson [16] evaluate every possible attribute set obtained from their 13 attributes to find the set that provides the best classification results. An exhaustive search is feasible on a small set of candidate attributes, but unrealistic on a larger set. Indeed, there are 2^n possible attribute sets for n candidate attributes.

6.2 Evaluation of the Sensitivity of an Attribute Set

Alaca et al. [3] rate fingerprinting attributes given properties that include the resource usage and the resistance to spoofing. They also model attackers according to different strategies and knowledge. The rating of the attributes mainly comes from estimations, and most of their spoof-resistant attributes are outside our boundaries on attribute choice. Indeed, we only consider the attributes collected via HTTP headers or JavaScript properties, that are accessible without permission and do not directly concern the user (e.g., IP address, geolocation) but rather his web browsing platform.

Laperdrix et al. [31] propose a challenge-response mechanism based on dynamic attributes which values also depend on provided instructions (e.g., the HTML5 canvas depends on drawing instructions). They identify various attacks that can be executed on an authentication mechanism that includes browser fingerprinting. The attacks notably include the submission of the most common fingerprints. We also consider the attacker that submits the most common fingerprints following his knowledge, and consider in addition a matching function to measure his reach in a realistic context (i.e., a fingerprint can spoof others that are similar).

7 CONCLUSION

In this study, we propose FPSelect, a framework for a verifier to tailor his fingerprinting probe by picking the attribute set that limits the sensitivity against an instantiated attacker, and reduces the usability cost. We formalize the Attribute Selection Problem that the verifier has to solve, show that it is a generalization of the Knapsack Problem, model the potential solutions as a lattice of attribute sets, and propose a greedy exploration algorithm to find a solution. We evaluate FPSelect on a real-life browser fingerprint dataset, and compare it with common attribute selection methods that rely on the entropy and the conditional entropy. The attribute sets found by FPSelect generate fingerprints that are 12 to 1,663 times smaller, 9 to 32,330 times faster to collect, and with 4 to 30 times less changing attributes between two observations, compared to the candidate attributes and on average. Compared to the baselines, the attribute sets found by FPSelect generate fingerprints that are up to 97 times smaller, are collected up to 3,361 times faster, and with up to 7.2 times less changing attributes between two observations, on average.

In future works, we will first extend our attack model with the attackers that possess targeted knowledge about users (e.g., the value of fixed attributes, components of their web environment). Indeed, the attacker that manages to infer the fingerprint distribution of small subpopulations (e.g., grouped by the operating system), and to link users to a subpopulation, would obtain a more skewed distribution which could help him to extend his reach. Moreover, the attacker that has the knowledge of previous challenges of dynamic attributes and the associated responses, can try to forge the response to an unseen challenge using other methods (e.g., image processing for the canvas). The study of the ability of these attackers, and the measure of the sensitivity against them, are let as future works. Second, the behavior of our framework on other experimental setups (browser population, dataset, measures, parameters) would be interesting, and is let as future works.

ACKNOWLEDGMENTS

We want to thank the anonymous reviewers for their usefull reviews; Benoît Baudry, David Gross-Amblard, Joris Duguépéroux, and Louis Béziaud for their valuable comments; and Alexandre Garel for his work on the experiment.

REFERENCES

[1] Mustafa Gunes Can Acar. 2017. Online Tracking Technologies and Web Privacy. https://lirias.kuleuven.be/retrieve/454271
[2] Nasser Mohammed Al-Fannah. 2017. One Leak will sink a Ship: WebRTC IP Address Leaks. In *International Carnahan Conference on Security Technology (ICCST)* (2017-10). 1–5. https://doi.org/10.1109/CCST.2017.8167801
[3] Furkan Alaca and P. C. van Oorschot. 2016. Device Fingerprinting for Augmenting Web Authentication: Classification and Analysis of Methods. In *Annual Conference on Computer Security Applications (ACSAC)* (2016-10). 289–301. https://doi.org/10.1145/2991079.2991091
[4] Nampoina Andriamilanto, Tristan Allard, and Gaëtan Le Guelvouit. 2021. "Guess Who?" Large-Scale Data-Centric Study of the Adequacy of Browser Fingerprints for Web Authentication. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)* (2021), Leonard Barolli, Aneta Poniszewska-Maranda, and Hyunhee Park (Eds.). 161–172. https://doi.org/10.1007/978-3-030-50399-4_16
[5] Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. 2020. A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication. (2020). https://arxiv.org/abs/2006.09511 under reviews.

[6] Mohammadreza Ashouri, Hooman Asadian, and Christian Hammer. 2018. Large-Scale Analysis of Sophisticated Web Browser Fingerprinting Scripts. (2018). https://hal.archives-ouvertes.fr/hal-01811691

[7] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. 2016. Disguised Chromium Browser: Robust Browser, Flash and Canvas Fingerprinting Protection. In *ACM Workshop on Privacy in the Electronic Society (WPES)* (2016). 37–46. https://doi.org/10.1145/2994620.2994621

[8] C. Blakemore, J. Redol, and M. Correia. 2016. Fingerprinting for Web Applications: From Devices to Related Groups. In *IEEE Trustcom/BigDataSE/ISPA* (2016-08). 144–151. https://doi.org/10.1109/TrustCom.2016.0057

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

[10] Joseph Bonneau. 2012. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *IEEE Symposium on Security and Privacy (S&P)* (2012-05). 538–552. https://doi.org/10.1109/SP.2012.49

[11] J. Bonneau, C. Herley, P. C. v Oorschot, and F. Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy (S&P)* (2012-05). 553–567. https://doi.org/10.1109/SP.2012.44

[12] Elie Bursztein, Artem Malyshev, Tadek Pietraszek, and Kurt Thomas. 2016. Picasso: Lightweight Device Class Fingerprinting for Web Clients. In *Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)* (2016-10-24). 93–102. https://doi.org/10.1145/2994459.2994467

[13] Peter Eckersley. 2010. How Unique is Your Web Browser?. In *International Conference on Privacy Enhancing Technologies (PETS)* (2010). 1–18. https://doi.org/10.1007/978-3-642-14527-8_1

[14] David Fifield and Serge Egelman. 2015. Fingerprinting Web Users Through Font Metrics. In *Financial Cryptography and Data Security (FC)* (2015), Rainer Böhme and Tatsuaki Okamoto (Eds.). 107–124. https://doi.org/10.1007/978-3-662-47854-7_7

[15] David Fifield and Mia Gil Epner. 2016. Fingerprintability of WebRTC. (2016). https://arxiv.org/abs/1605.08805

[16] Erik Flood and Joel Karlsson. 2012. Browser Fingerprinting. https://hdl.handle.net/20.500.12380/163728

[17] Maximilian Golla, Theodor Schnitzler, and Markus Dürmuth. 2018. "Will Any Password Do?" Exploring Rate-Limiting on the Web. In *USENIX Symposium on Usable Privacy and Security (SOUPS)* (2018-08-12).

[18] Gábor György Gulyás, Gergely Acs, and Claude Castelluccia. 2016. Near-Optimal Fingerprinting with Constraints. 2016, 4 (2016), 470–487. https://doi.org/10.1515/popets-2016-0051

[19] 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 *The Web Conference (TheWebConf)* (2018-04). https://doi.org/10.1145/3178876.3186097

[20] Weili Han, Zhigong Li, Minyue Ni, Guofei Gu, and Wenyuan Xu. 2018. Shadow Attacks Based on Password Reuses: A Quantitative Empirical Analysis. 15, 2 (2018), 309–320. https://doi.org/10.1109/TDSC.2016.2568187

[21] Eiji Hayashi, Rachna Dhamija, Nicolas Christin, and Adrian Perrig. 2008. Use your Illusion: Secure Authentication Usable Anywhere. In *Symposium on Usable Privacy and Security (SOUPS)* (2008). 35–45. https://doi.org/10.1145/1408664.1408670

[22] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. 1998. Support Vector Machines. 13, 4 (1998), 18–28. https://doi.org/10.1109/5254.708428

[23] Peter Hraška. 2018. Browser Fingerprinting. https://virpo.sk/browser-fingerprinting-hraska-diploma-thesis.pdf

[24] Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing* (2 ed.). Pearson. 23–27 pages.

[25] Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2Nd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 325–326 pages.

[26] Nian-hua KANG, Ming-zhi CHEN, Ying-yan FENG, Wei-ning LIN, Chuan-bao LIU, and Guang-yao LI. 2017. Zero-Permission Mobile Device Identification Based on the Similarity of Browser Fingerprints. In *International Conference on Computer Science and Technology (CST)* (2017-07-31). https://doi.org/10.12783/dtcse/cst2017/12531

[27] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *Network and Distributed System Security Symposium (NDSS)* (2020). https://doi.org/10.14722/ndss.2020.24383

[28] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack Problems.* Springer-Verlag. https://www.springer.com/gp/book/9783540402862

[29] Amin Faiz Khademi, Mohammad Zulkernine, and Komminist Weldemariam. 2015. An Empirical Evaluation of Web-Based Fingerprinting. 32, 4 (2015), 46–52. https://doi.org/10.1109/MS.2015.77

[30] Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling. 2016. Fingerprinting Mobile Devices Using Personalized Configurations. 2016, 1 (2016). https://doi.org/10.1515/popets-2015-0027

[31] Pierre Laperdrix, Gildas Avoine, Benoit Baudry, and Nick Nikiforakis. 2019. Morellian Analysis for Browsers: Making Web Authentication Stronger With Canvas Fingerprinting. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (2019-06). 43–66. https://doi.org/10.1007/978-3-030-22038-9_3

[32] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2015. Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification. In *IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (2015-05). 98–108. https://doi.org/10.1109/SEAMS.2015.18

[33] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In *IEEE Symposium on Security and Privacy (S&P)* (2016-05). 878–894. https://doi.org/10.1109/SP.2016.57

[34] Paul Marks. 2020. Dark Web's Doppelgängers Aim to Dupe Antifraud Systems. 63, 2 (2020), 16–18. https://doi.org/10.1145/3374878

[35] João Pedro Figueiredo Correia Rijo Mendes. 2011. noPhish – Anti-phishing System using Browser Fingerprinting. https://estagios.dei.uc.pt/cursos/mei/relatorios-de-estagio/?id=279

[36] Grzergor Milka. 2018. Anatomy of Account Takeover. In *Enigma* (2018). https://www.usenix.org/node/208154

[37] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. 2011. Fingerprinting Information in JavaScript Implementations. In *Proceedings of W2SP* (2011-05), Helen Wang (Ed.), Vol. 2.

[38] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. , 12 pages. https://www.ieee-security.org/TC/W2SP/2012/papers/w2sp12-final4.pdf

[39] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, Edgar Weippl, and FC Wien. 2013. Fast and Reliable Browser Identification with Javascript Engine Fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)* (2013), Vol. 5.

[40] Gabi Nakibly, Gilad Shelef, and Shiran Yudilevich. 2015. Hardware Fingerprinting Using HTML5. (2015). https://arxiv.org/abs/1503.01408

[41] Panagiotis Papadopoulos, Panagiotis Ilia, Michalis Polychronakis, Evangelos P. Markatos, Sotiris Ioannidis, and Giorgos Vasiliadis. 2019. Master of Web Puppets: Abusing Web Browsers for Persistent and Stealthy Computation. In *Network and Distributed System Security Symposium (NDSS)* (2019-02). https://doi.org/10.14722/ndss.2019.23070

[42] Thanasis Petsas, Giorgos Tsirantonakis, Elias Athanasopoulos, and Sotiris Ioannidis. 2015. Two-factor Authentication: Is the World Ready? Quantifying 2FA Adoption. In *European Workshop on System Security (EuroSec)* (2015-04-21). 1–7. https://doi.org/10.1145/2751323.2751327

[43] Davy Preuveneers and Wouter Joosen. 2015. SmartAuth: Dynamic Context Fingerprinting for Continuous User Authentication. In *Annual ACM Symposium on Applied Computing (SAC)* (2015). 2185–2191. https://doi.org/10.1145/2695664.2695908

[44] Gaston Pugliese, Christian Riess, Freya Gassmann, and Zinaida Benenson. 2020. Long-Term Observation on Browser Fingerprinting: Users' Trackability and Perspective. 2020, 2 (2020), 558–577. https://doi.org/10.2478/popets-2020-0041

[45] Jordan S. Queiroz and Eduardo L. Feitosa. 2019. A Web Browser Fingerprinting Method Based on the Web Audio API. (2019). https://doi.org/10.1093/comjnl/bxy146

[46] Florentin Rochet, Kyriakos Efthymiadis, François Koeune, and Olivier Pereira. 2019. SWAT: Seamless Web Authentication Technology. In *The Web Conference (TheWebConf)* (2019-05). 1579–1589. https://doi.org/10.1145/3308558.3313637

[47] Julian F. Reschke Roy T. Fielding. 2014. *RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* https://tools.ietf.org/html/rfc7231#section-5.5.3 accessed 2020-06-30.

[48] T. Saito, K. Yasuda, T. Ishikawa, R. Hosoi, K. Takahashi, Y. Chen, and M. Zalasiński. 2016. Estimating CPU Features by Browser Fingerprinting. In *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)* (2016-07). 587–592. https://doi.org/10.1109/IMIS.2016.108

[49] Takamichi Saito, Koki Yasuda, Kazuhisa Tanabe, and Kazushi Takahashi. 2017. Web Browser Tampering: Inspecting CPU Features from Side-Channel Information. In *International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)* (2017-11-08). 392–403. https://doi.org/10.1007/978-3-319-69811-3_36

[50] Rachel Schutt and Cathy O'Neil. 2014. *Doing data science: Straight talk from the frontline.* O'Reilly. 181–182 pages.

[51] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. 2017. Discovering Browser Extensions via Web Accessible Resources. In *ACM Conference on Data and Application Security and Privacy (CODASPY)* (2017). 329–336. https://doi.org/10.1145/3029806.3029820

[52] Jesus Solano, Luis Camacho, Alejandro Correa, Claudio Deiro, Javier Vargas, and Martín Ochoa. 2019. Risk-Based Static Authentication in Web Applications with

Behavioral Biometrics and Session Context Analytics. In *Applied Cryptography and Network Security Workshops (ACNS)* (2019), Jianying Zhou, Robert Deng, Zhou Li, Suryadipta Majumdar, Weizhi Meng, Lingyu Wang, and Kehuan Zhang (Eds.). 3–23. https://doi.org/10.1007/978-3-030-29729-9_1

[53] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2015. Mobile Device Fingerprinting Considered Harmful for Risk-based Authentication. In *European Workshop on System Security (EuroSec)* (2015). 6:1–6:6. https://doi.org/10.1145/2751323.2751329

[54] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2017. Leveraging Battery Usage from Mobile Devices for Active Authentication. (2017). https://doi.org/10.1155/2017/1367064

[55] Oleksii Starov and Nick Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *IEEE Symposium on Security & Privacy (S&P)* (2017-05-24). 941–956. https://doi.org/10.1109/SP.2017.18

[56] Kazuhisa Tanabe, Ryohei Hosoya, and Takamichi Saito. 2018. Combining Features in Browser Fingerprinting. In *Advances on Broadband and Wireless Computing, Communication and Applications (BWCCA)* (2018), Leonard Barolli, Fang-Yie Leu, Tomoya Enokido, and Hsing-Chung Chen (Eds.). 671–681. https://doi.org/10.1007/978-3-030-02613-4_60

[57] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, Daniel Margolis, Vern Paxson, and Elie Bursztein. 2017. Data Breaches, Phishing, or Malware? Understanding the Risks of Stolen Credentials. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2017-10-30). 1421–1434. https://doi.org/10.1145/3133956.3134067

[58] T. Unger, M. Mulazzani, D. Frühwirt, M. Huber, S. Schrittwieser, and E. Weippl. 2013. SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting. In *International Conference on Availability, Reliability and Security (ARES)* (2013-09). 255–261. https://doi.org/10.1109/ARES.2013.33

[59] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *IEEE Symposium on Security and Privacy (S&P)* (2018-05-21). 728–741. https://doi.org/10.1109/sp.2018.00008

[60] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. 2016. Targeted Online Password Guessing: An Underestimated Threat. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2016-10-24). 1242–1254. https://doi.org/10.1145/2976749.2978339

[61] Matt Weir, Sudhir Aggarwal, Breno de Medeiros, and Bill Glodek. 2009. Password Cracking Using Probabilistic Context-Free Grammars. In *IEEE Symposium on Security and Privacy (S&P)* (2009-05). 391–405. https://doi.org/10.1109/SP.2009.8

[62] Wenjia Wu, Jianan Wu, Yanhao Wang, Zhen Ling, and Ming Yang. 2016. Efficient Fingerprinting-Based Android Device Identification with Zero-Permission Identifiers. 4 (2016), 8073–8083. https://doi.org/10.1109/ACCESS.2016.2626395

A DEMONSTRATIONS OF MEASURES MONOTONICITY

A.1 Monotonicity of the Illustrative Sensitivity

Theorem A.1. *Considering the same limit on the number of submissions β, the mapping from users to their browser fingerprint $\mathcal{M}$, the probability mass function p, and the set of matching functions Φ. For any couple of attribute sets C_k and C_l so that $C_k \subset C_l$, we have $\mathrm{s}(C_k) \geq \mathrm{s}(C_l)$ when measuring the sensitivity using Algorithm 2.*

Proof Sketch. We focus on a fingerprint $f \in F_{C_l}$ when considering the attribute set C_l, and the dictionary V used to attack f. We project f to the attribute set C_k to obtain the fingerprint $g \in F_{C_k}$. This fingerprint g can be attacked using the dictionary W composed of the fingerprints of V projected to C_k. As the matching function works in an attribute basis, if a fingerprint $h \in V$ matches with f, we have $f[a] \approx^a h[a] : \forall a \in C_l$ that is true. As C_k is a subset of C_l, we also have $f[a] \approx^a h[a] : \forall a \in C_k$ that is true. The projection of h to C_k then spoofs g, hence the fingerprints spoofed when considering C_l are also spoofed when considering C_k. The sensitivity when considering C_k is therefore at least equal to that when considering C_l.

When projecting the fingerprints of the attack dictionary to the attribute set C_k, some of them can come to the same fingerprint. In which case, more fingerprints can be added to the dictionary until reaching the submission limit β. This can lead to more spoofed fingerprints and impersonated users. The sensitivity when considering C_k can be higher than when considering C_l.

For any attribute sets C_k and C_l so that $C_k \subset C_l$, we then have $\mathrm{s}(C_k) \geq \mathrm{s}(C_l)$ when measuring the sensitivity using Algorithm 2. □

A.2 Monotonicity of the Illustrative Usability Cost

Theorem A.2. *For a given fingerprint dataset D, and attribute sets C_k and C_l so that $C_k \subset C_l$, we have $\mathrm{cost}(C_k, D) < \mathrm{cost}(C_l, D)$.*

Proof. We consider C_i and C_j two attribute sets, so that they differ by the attribute a_d such that $C_j = C_i \cup \{a_d\}$. We consider the fingerprint dataset D from which the measures are obtained.

The memory cost of C_j is strictly greater than the memory cost of C_i. As C_j and C_i differ by the attribute a_d, we have

$$\mathrm{mem}(C_j, D) = \mathrm{mem}(C_i, D) + \frac{\sum_{(b,f)\in D} \mathrm{size}(f[a_d])}{\mathrm{card}(D)} \tag{5}$$

The size of the attribute a_d is strictly positive, hence we have the inequality $\mathrm{mem}(C_j, D) > \mathrm{mem}(C_i, D)$.

The temporal cost of C_j is greater than or equal to the temporal cost of C_i, as adding an attribute cannot reduce the collection time. The attribute a_d can be sequential or asynchronous, and can take identical or more time than the current longest attribute of C_i. Below, we explore all these cases. (1) If a_d is asynchronous, we have the following cases: (a) a_d takes less time than the longest asynchronous attribute a_l, then the collection time is either that of a_l or the total of the sequential attributes, so a_d does not influence the collection time and $\mathrm{time}(C_j, D) = \mathrm{time}(C_i, D)$, (b) a_d takes more time than the longest asynchronous attribute, but less or equal to the total of the sequential attributes, then the maximum is the total of the sequential attributes and $\mathrm{time}(C_j, D) = \mathrm{time}(C_i, D)$, (c) a_d takes more time than both the longest asynchronous attribute and the total of the sequential attributes, then the collection time is that of a_d, and $\mathrm{time}(C_j, D) > \mathrm{time}(C_i, D)$. (2) If a_d is sequential, we have the following cases: (a) a_d increases the total collection time of the sequential attributes, but the total stays below that of the longest asynchronous attribute, then we have the equality $\mathrm{time}(C_j, D) = \mathrm{time}(C_i, D)$, (b) a_d increases the total collection time of the sequential attributes, which is then higher than that of the longest asynchronous attribute, then $\mathrm{time}(C_j, D) > \mathrm{time}(C_i, D)$[23]. These are all the possible cases, hence $\mathrm{time}(C_j, D) \geq \mathrm{time}(C_i, D)$.

The instability cost of C_j is greater than or equal to that of C_i, as either a_d is completely stable and $\mathrm{ins}(C_j, D) = \mathrm{ins}(C_i, D)$, otherwise a_d is unstable and $\mathrm{ins}(C_j, D) > \mathrm{ins}(C_i, D)$. We then have $\mathrm{ins}(C_j, D) \geq \mathrm{ins}(C_i, D)$.

As the cost weight vector γ is composed of strictly positive numbers, the cost of C_j is therefore strictly higher than the cost of C_i

[23] Either the total collection time of the sequential attributes of C_i does not exceed that of its longest asynchronous attribute, and adding a_d results in the total collection time of the sequential attributes surpassing that of the longest asynchronous attribute. In this case, $\mathrm{time}(C_j, D) > \mathrm{time}(C_i, D)$. Either the total collection time of the sequential attributes of C_i exceeds that of its longest asynchronous attribute. As a_d increases the total collection time of the sequential attributes, we have $\mathrm{time}(C_j, D) > \mathrm{time}(C_i, D)$.

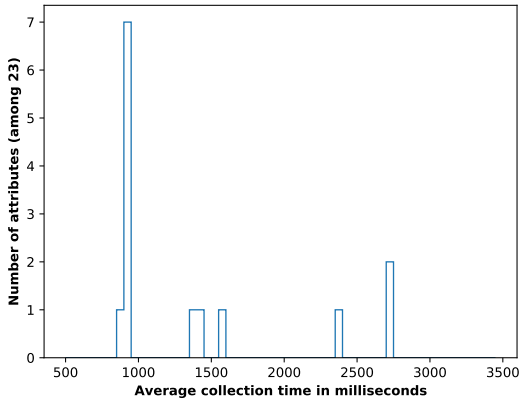


Figure 7: Distribution of the average collection time among the 23 asynchronous attributes.

due to the memory cost. Recursively, it holds for any C_k and C_l so that $C_k \subset C_l$, hence the cost is monotonic. For any fingerprint dataset D, and attribute sets C_k and C_l so that $C_k \subset C_l$, we have $\text{cost}(C_k, D) < \text{cost}(C_l, D)$. □

B USABILITY COST OF ATTRIBUTES

In this appendix, we discuss the distribution of the three usability cost dimensions among the attributes. For more insight into the fingerprints and the attributes of the dataset of this study, we refer to the study of Andriamilanto et al. [5] which includes an analysis of this dataset.

Figure 7 presents the distribution of the average collection time among the 23 asynchronous attributes. These attributes comprise the extension detection methods that require to wait for the web page to render [51, 55], heavy processes like the WebRTC fingerprinting method [2, 15] that creates dummy connections, and the audio fingerprinting methods [45]. Figure 8 presents the distribution of the average collection time among the 173 sequential attributes. Only 9 attributes take more than 25 milliseconds on average to collect, among which we retrieve the six canvases [33, 38], the list of WebGL extensions, the list of available streaming codecs, and the string representation of a specific date. The sequential and asynchronous attributes do not sum to the 253 candidate attributes. We do not show the HTTP headers that are collected instantly, which have a null collection time. Moreover, we only display the collection time of the attributes source of the extracted attributes, as they have the same collection time. Indeed, the source attributes are the one that are actually collected, and the extracted attributes are inferred from them.

Figure 9 presents the distribution of the average storage size among the 253 attributes. We have 29 attributes that weigh more than 50 bytes on average. They are the list attributes (e.g., list of plugins), the verbose properties (e.g., the `UserAgent`), and the string representation of the hashed canvases.

Figure 10 presents the distribution of the average instability of the 253 attributes. Only 16 attributes have more than 0.05 changes per observation on average. They comprise the attributes related to the screen resolution, the size of the browser window, the canvases, the experimental WebRTC fingerprinting method, the list of speech synthesis voices, the `Cache-Control` HTTP header, and the attribute that stores the HTTP headers that are not stored in a dedicated attribute.

C ATTRIBUTES SELECTED BY THE ATTRIBUTE SELECTION FRAMEWORK

In this appendix, we provide the list of the 21 attributes that are selected by the attribute selection framework. Table 4 lists the selected attributes. We name them according to the same nomenclature as [5]. We denote *N* the `navigator` object, *S* the `screen` object, *W* the `window` object, and *A* an initialized *Audio Context*. Finally, we denote *WG* an initialized *WebGL Context*, and *WM* the `WG.MAX_` prefix. To get the WG.[...].UNMASKED_RENDERER_WEBGL attribute, we first get an identifier named `id` from the unmasked property of the `getExtension('WEBGL_debug_renderer_info')` object, and then get the actual value by calling `getParameter(id)`. We use square brackets so that `A.[B, C]` means that the property is accessed through `A.B` or `A.C`.

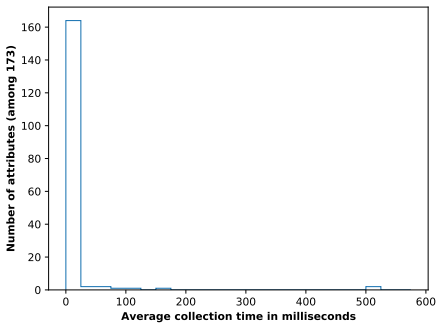


Figure 8: Distribution of the average collection time among the 173 sequential attributes.

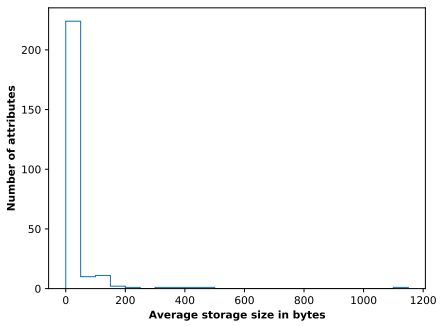


Figure 9: Distribution of the average storage size among the attributes.

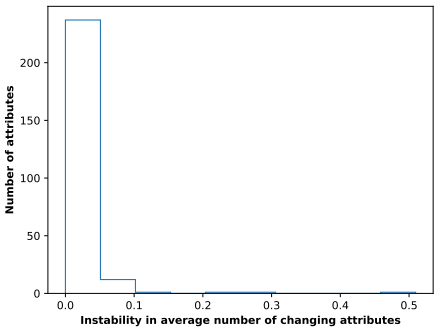


Figure 10: Distribution of the average instability among the attributes.

Selected Attribute	$k=1$									$k=3$								
	$\beta=1$				$\beta=4$			$\beta=16$		$\beta=1$				$\beta=4$			$\beta=16$	
	$\alpha=0.001$	$\alpha=0.005$	$\alpha=0.015$	$\alpha=0.025$	$\alpha=0.005$	$\alpha=0.015$	$\alpha=0.025$	$\alpha=0.015$	$\alpha=0.025$	$\alpha=0.001$	$\alpha=0.005$	$\alpha=0.015$	$\alpha=0.025$	$\alpha=0.005$	$\alpha=0.015$	$\alpha=0.025$	$\alpha=0.015$	$\alpha=0.025$
W.innerHeight	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
N.hardwareConcurrency	●		●		●	●		●	●			●		●	●	●	●	
WG.[...].UNMASKED_RENDERER_WEBGL		●				●	●	●			●				●		●	●
N.appVersion	●				●			●		●				●			●	●
HTML5 canvas inspired by AmIUnique (PNG)	●				●			●	●	●							●	●
N.connection.type	●				●			●		●				●			●	
N.plugins	●							●	●	●							●	
[[N, W].doNotTrack, N.msDoNotTrack]		●		●	●						●							
Accept-Encoding HTTP header	●									●				●				
Height of first bounding box	●									●								
Origin of a created `div`					●									●				
W.ontouchstart support													●			●		
W.[performance, console].jsHeapSizeLimit					●													
S.width										●								
W.openDatabase support													●					
WM.COMBINED_TEXTURE_IMAGE_UNITS														●				
N.platform														●				
HTML5 canvas similar to Morellian (PNG)														●				
Accept-Language HTTP header														●				
WM.CUBE_MAP_TEXTURE_SIZE																●		
A.sampleRate																●		

Table 4: The attributes selected by the attribute selection framework for each experimentation setup. We denote k the number of explored paths, β the number of fingerprint submissions, and α the sensitivity threshold.